UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE INGENIERÍA ELECTRÓNICA Y ELÉCTRICA

UNIDAD DE POSTGRADO

Sistema multi-agente para la medición de los parámetros de performance de una red de datos

TESIS

para optar el grado académico de Magíster en Telecomunicaciones con Mención en Redes y Servicios de Banda Ancha

AUTOR

Eladio Llamoga Sánchez

Lima-Perú

2009

AGRADECIMIENTOS

Es una excelente oportunidad para agradecer a todas las personas que, de una u otra manera han contribuido para que el desarrollo de este proyecto sea una realidad.

Primero agradecer a Dios, por haberme mostrado su ayuda, y valor en los momentos difíciles, durante el desarrollo de esta investigación.

A mi orientador Msc. José Luis Muñoz Meza, docente de la Escuela de Post-grado, por su exigencia y siempre crítica constructiva que me ayudó a mejorar y completar este trabajo.

Cabe también mencionar mi agradecimiento a todos los participantes del Fórum de discusiones de AdventNet, por su colaboración en responder a algunas de las dudas durante ciertas fases de la evolución de este trabajo.

Por último, gracias a mi familia, en especial a mis padres María y Manuel por su comprensión y cariño, y a mis hermanos Francisca, Alejandro y Segundo, que me han servido para no rendirme en el prolongado y en ocasiones difíciles periodo de tiempo que ha conllevado realizar esta investigación.

A todos un sincero agradecimiento

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS

ACL	Agent Communication Language
ADK	Agent Development Kit
AID	Agent Identifier
AMS	Agent Management System
ASN.1	Abstract Syntax Notation One
ANSI	American National Standards Institute
API	Application Programming Interface
BER	Basic Encoding Rules
CCITT	Consultative Committee for International Telegraph and Telephone
CIM	Common Information Model
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CMISE	Common Management Information Service Element
DISMAN	Distributed Management
DMTF	Distributed Management Task Force
DF	Directory Facilitator
FIPA	Foundation for Intelligent Physical Agents
GIOP	General Inter-ORB Protocol
GUI	Graphical User Interface
GMDO	Guidelines for Definition of Management Objects
HTML	Hyper Text Markup Language
IETF	Internet Engineering Task Force
IAB	Internet Architecture Board
IDL	Interface Definition Language
IMTP	Internal Message Transport Protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
ISO	International Standards Organization
ITU	International Telecommunication Union
JVM	Java Virtual Machine

JDBC	Java Data Base Connectivity
KQML	Knowledge Query Manipulation Language
KIF	Knowledge Interchange Format
GPL	General Public License
MAF	Mobil Agent Framework
MASIF	Mobil Agent System Interoperable Facilities
MIB	Management Information Base
MTP	Message Transport Protocol
NMS	Network Management Station
OID	Object Identifier
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open System Interconnection
PDU	Packet Data Unit
RPC	Remote Procedure Calls
RFC	Request For Comments
RMI	Remote Method Invocation
SAS	SNMP Applet Server
SL	Semantic Language
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SO	System Operating
TCL	Tool Command Language
TCP	Transmission Control Protocol
TMN	Telecommunication Management Network
UDP	User Datagram Protocol
UML	Unified Modeling Language
USM	User Security Model
VACM	View Access Control Model
WAP	Wireless Application Protocol
WBEM	Web-Based Enterprise Management
XML	Extensible Markup Language

RESUMEN

Hoy en día, la utilización de redes de comunicaciones heterogéneas se ha vuelto en un desafío constante, debido a sus complejidades y crecimiento acelerado, tornándose en un factor crítico de sucesos en muchas organizaciones.

Durante la última década, diversas arquitecturas de gestión de red han sido desarrolladas y estandarizadas permitiendo el control y monitorización de los diversos indicadores de redes de telecomunicaciones tales como: la utilización de la interfaz, tasa de error, rendimiento y el retardo extremo a extremo, afectando de manera sustancial la calidad de operación de toda red, percibida directamente (p.e en el servicio VoIP) o indirectamente (p.e. capacidad de transferencia de los protocolos de transporte) por los usuarios finales. Todo esto basados en la interconexión de redes de distintas tecnologías, operadas por distintos proveedores y con el paradigma de servicio best-effort.

En telecomunicaciones, el término “Calidad de Servicio” o QoS, hace referencia precisamente a las técnicas y procedimientos utilizados para dar un tratamiento preferente a unas clases de tráfico o flujos frente a otras, con el objetivo de cumplir unos requisitos mínimos de performance, identificados por parámetros cuantitativos que deben ser respetados estrictamente para que se logre percibir una adecuada calidad subjetiva por parte de los usuarios, particularmente en las comunicaciones multimediales, lo que da lugar al término “Calidad de Experiencia” o QoE como una nueva forma de valoración de la calidad final ofrecida por las empresas proveedoras de servicio.

Con estos antecedentes, la presente investigación explica una nueva forma de calcular estos parámetros, usando un prototipo de aplicación de gestión, de modo que sobre la base de estas mediciones, puedan hacerse los ajustes necesarios dentro de la red y los administradores puedan identificar congestión o potenciales eventos indeseables en la red.

De acuerdo a investigaciones bien documentadas sobre este tema, en donde la gestión llegar a ser un factor importante, el paradigma tradicional de arquitectura centralizada es particularmente ineficiente durante periodos de alta congestión, debido al incremento de las interacciones entre la estación central y los dispositivos gestionados. De esta manera,

esta investigación pretende suplir en determinada medida esta deficiencia, proponiendo un sistema de gestión de red, basado en agente móviles denominado "AgenteMobilSNMP – AMSNMP" aplicado dentro de un entorno distribuido para llevar a cabo el proceso de gestión y la recolección de los resultados.

Esta tesis, está basada en una investigación cuantitativa, que compara ambos esquemas de gestión para cuantificar y validar esta propuesta, permitiendo una mejor evaluación de los métodos aplicados para encontrar valores de los parámetros performance, en cada uno de los procesos desarrollados y así obtener información comparable, pertinente y actualizada en términos del tiempo y tráfico ocasionados.

ABSTRACT

Nowadays, the use of heterogeneous communications networks has become a constant challenger because of its complexities and accelerated growth, becoming a critical factor in many organizations.

During the last decade, many network management architectures have been developed and standardized allowing the control and monitoring of several telecommunications networks indicators such as: the use of the interface, error rate, throughput, and the end-to-end delay, significantly affecting the quality of operation of the whole network, which is perceived directly (e.g. in VoIP service) or indirectly (e.g. transference capacity of the transport protocols) by the end users. All this made through the interconnection of networks of different technologies, operated by different suppliers and generally with the "best-effort" service paradigm.

In telecommunications, the term "Quality of Service" or QoS, especially refers to techniques and procedures used to give preferential treatment to some class or traffic flows, over others, to meet minimum standards requirements of performance, identified by quantitative parameters, that must be strictly respected such that a suitable subjective quality is perceived by users, particularly in multimedia communications, where the term "Quality of Experience" or QoE as a new way of valuing the final quality offered by the companies service providers.

With this background, the present document explains a new way to calculate these parameters, using a prototype management application so that on the basis of these measurements, adjustments can be made in the network and the administrators can be able to identify congestion trends and potentials undesirable events for the operation of the network.

According to well documented researches on this subject, the centralized paradigm used by the traditional SNMP management architecture, among other aspects, where the central station (or management console) obtains the results of each managed station, one at a time, it is too restricted, because the amount of traffic up to this architecture cause, account to its, a proposing management system based on mobile agents called "AgenteMobilSNMP – AMSNMP, applied within a distributed environment, where the

agent is transported throughout all the entities to manage in order to carry out the process management and data collection.

This thesis, a quantitative-based research, compares both management schemes in order to quantify and validate its proposal and allowing a better evaluation of the methods applied to find the network parameters values, in each of these developed processes and thus to obtain comparable, pertinent and updated data in terms of load processing and traffic offered to the end users.

INTRODUCCION

La evolución de las aplicaciones de software ha venido de la mano con su integración al mundo de las telecomunicaciones, debido en gran medida al desarrollo de las redes de banda ancha y especialmente al auge de la tecnología de Internet.

Los procesos de planificación, diseño e implementación de toda red de comunicaciones de datos requieren de un estudio profundo y minucioso, teniendo en cuenta sus condiciones de operatividad y funcionalidad, esto es, de acuerdo a especificaciones técnicas, utilización y a las aplicaciones que se hacen de ellas. El cumplimiento o no de estas condiciones se reflejarán de manera precisa a través de indicadores que en conjunto conforman la "calidad de servicio" ofrecida por la red.

En otras palabras, es necesario efectuar una gestión de red de forma que se pueda controlar y supervisar de manera adecuada con el objetivo de maximizar su eficiencia y productividad.

Actualmente existen arquitecturas de gestión, en torno a modelos y técnicas que permiten contribuir mejor las operaciones de gestión, dentro de las cuales se pueden mencionar: La gestión OSI (*Open Systems Interconnection*), gestión SNMP (*Simple Network Management Protocol*), y gestión TMN (*Telecommunication Management Network*), que explicaremos en el capitulo2.

Por otro lado la mayoría de estas arquitecturas siguen el paradigma de la estructura centralizada que, a pesar de ser aceptable desde hace muchos años, se revela que es insuficiente para lidiar con el aumento de dimensión y de tráfico. Durante varios años, surgieron varias propuestas, modelos y arquitecturas de distribución de gestión que procuran eliminar estos problemas de centralización. Es bajo este contexto que han surgido los denominados sistemas distribuidos, en los cuales un sistema no depende de solo un punto, si no que las tareas de gestión están distribuidas por toda la red, lo que ha permitido el desarrollo de algunas arquitecturas como: CORBA, JAVA-RMI y la tecnología de agentes móviles.

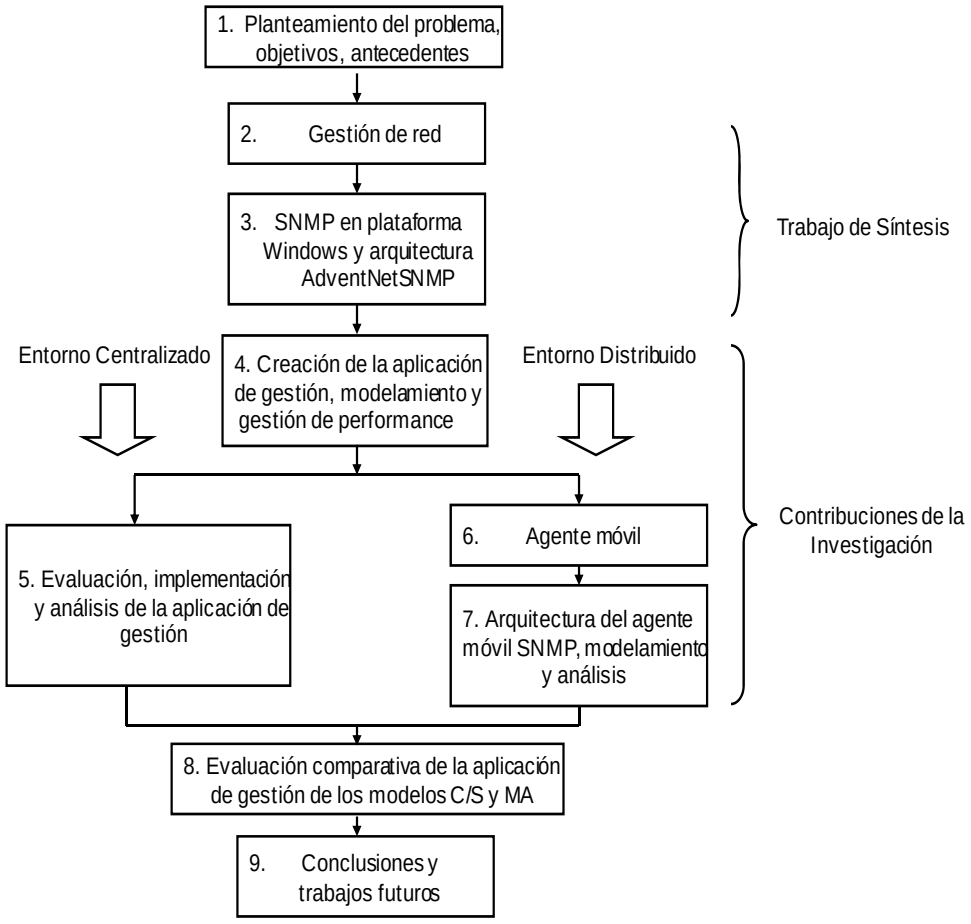
En este trabajo de investigación, se desarrollará una aplicación de sistema de gestión de red teniendo como base al agente SNMP, denominado "SNMPAgentTest", el cual será implementado primero dentro de un entorno

centralizado, y luego en un entorno distribuido mediante otra aplicación (agente móvil) denominado “AgenteMobilSNMP - AMSNMP”, que se encargará de transportar dicha aplicación de gestión a los diferentes elementos de la red, y retornar al final con los resultados. De esta manera, poder así establecer las semejanzas o diferencias, y seleccionar el método adecuado a la hora de desplegar todo proceso de gestión.

Ambas aplicaciones han sido desarrolladas en Java, utilizando librerías y productos de softwares adicionales, para poder evaluar ambos enfoques: centralizado versus distribuido y así llegar a conclusiones respecto a que sistema es el más adecuado para aliviar la carga de trabajo, reducción de tráfico y consumir menos recursos en la red.

ORGANIZACIÓN DE LA TESIS

La tesis está estructurada de la siguiente manera:



Estructura de la tesis de investigación

Este trabajo de investigación se divide en nueve capítulos, los cuales podemos describir:
En el primer capítulo se describe la realidad problemática, los objetivos, y los antecedentes que conllevaron a esta investigación.

En el segundo capítulo se describe los modelos de gestión de red, cubriendo los modelos de gestión OSI, gestión SNMP, y la gestión TMN, haciendo un resumen de cada uno de ellos y la importancia de su implementación, dentro de cada uno de sus entornos.

En el tercer capítulo está dedicado a la plataforma Windows desde el punto de vista de gestión SNMP y a la arquitectura AdventNet SNMP y su utilización en el desarrollo de aplicaciones basadas en el modelo de gestión SNMP.

En el cuarto capítulo iniciamos con el desarrollo de la aplicación misma, el procedimiento llevado a cabo y los componentes o elementos que intervienen para dicho desarrollo. Luego se establecen los parámetros de performance y los objetos que intervendrán en las operaciones de gestión.

En el quinto capítulo se establecen las ecuaciones matemáticas de la aplicación de gestión, así como de los detalles de su implementación, ejecución y la obtención de sus resultados, mostrados estadísticamente.

En el capitulo seis se describe a la tecnología móvil, su importancia a la hora de gestionar, presentando a las diversas plataformas que existen en el mercado, así como a la plataforma JADE utilizada para el desarrollo de agentes móviles.

En el capitulo siete establecemos una arquitectura de agente móvil dentro del cual se construye el agente móvil SNMP que se encargará de transportar a la aplicación. Luego se establecen las ecuaciones matemáticas de este modelo de gestión, así como de su implementación y ejecución, para luego mostrar los resultados y muestrearlos en tablas gráficamente.

En el capitulo ocho se hace un análisis comparativo de ambos modelos de gestión, realizando la interpretación de las ecuaciones establecidas en los capítulos cinco y siete.

En el capitulo nueve se incluye las conclusiones de las principales consecuencias a las que se han arribado del proceso de investigación, al realizar la comparación de los resultados de ambos procedimientos. Luego se presentan los posibles trabajos futuros que puede conllevar la investigación realizada.

CAPITULO 1 Planteamiento del Problema, Objetivos, Antecedentes

1.1 PLANTEAMIENTO DEL PROBLEMA

El uso de las tecnologías IP están creciendo rápidamente en el mundo de las telecomunicaciones, y uno de los problemas principales consiste en gestionar los recursos de red de acuerdo a los nuevos servicios ofrecidos, en donde los elementos de red deben ser gestionados, monitoreados y reconfigurados dinámicamente para así permitir a los administradores identificar congestión o posible congestión en toda la red.

Por otro lado, la no adecuada calidad de servicio (QoS) que proporcionan algunos elementos de red hacen que ocasionen un comportamiento ineficiente, en los cuales unos son de mayor consideración que otros, que inclusive ocasionan la caída de la misma.

Todo esto nos lleva a hacer un estudio y análisis de los diferentes determinantes que perturban dicha red, como por ejemplo, en el proceso de tráfico que puede ser descrita en términos de características de un número de objetos, que incluyen paquetes, flujos, sesiones, retardos, y conexiones adecuadas en la escala de tiempo y de sus variaciones estadísticas relevantes [Roberts01], por lo que estos parámetros deben ser comprobados por los operadores de red y prestadores de

servicio, los cuales deben reportarse en un formato que sea comparable y publicado de forma que sea entendible por los usuarios finales.

Otro de los factores que hay que tener en cuenta es la dispersión geográfica y la cantidad de elementos de red asociados que pueden transportar grandes cantidades de información, y el hecho asociado al paradigma tradicional centralizado, pondrá bajo ciertas condiciones provocar inaccesibilidad a la estación de gestión debido a la sobrecarga de procesamiento y de tráfico y por lo tanto, a una eventual caída de la red. Actualmente las organizaciones han puesto su atención en sistemas que acepten modelos de gestión distribuida, ya que estos generan menos congestión, consumen menos recursos, y por lo tanto, generan menos tráfico de paquetes en la red.

1.2 OBJETIVO GENERAL

Determinar la relación de un sistema multi-agente para la medición de los parámetros de performance, aplicado a una red de datos dentro de un sistema de gestión de red.

1.2.1 OBJETIVOS ESPECIFICOS

- Desarrollar una aplicación de gestión de red, y establecer los parámetros de performance.
- Ejecutar la aplicación basado en el modelo de gestión SNMP dentro de un entorno centralizado.
- Establecer una nueva arquitectura de gestión para el desarrollo de un agente móvil de gestión, cuya función es llevar el código de la aplicación de gestión a los diferentes elementos de la red, esto dentro de un entorno distribuido.
- Establecer las semejanzas ó diferencias de ambos procesos de gestión.

1.3 METODOLOGIA

La metodología utilizada en el desarrollo de esta investigación es de tipo cuantitativo-deductivo y experimental, cuya obtención de datos apoyados en

escalas numéricas permitirán un tratamiento estadístico de los diferentes niveles de cuantificación, los cuales serán sometidos a contrastación.

La aplicación de gestión desarrollada en esta investigación fue construida teniendo como base la arquitectura de gestión AdventNet, cuyas herramientas permitieron realizar la evaluación de los parámetros de performance y la obtención de resultados, y observar el posible tráfico en la red.

El método de evaluación de la aplicación podemos enfocarlo desde dos puntos de vista de modelos: el modelo centralizado y el modelo distribuido.

Desde el enfoque de modelo de gestión centralizado, la estación de gestión ejecutará las operaciones de gestión mediante polling a cada una de las entidades de red o estaciones a gestionar.

Desde el enfoque de modelo de gestión distribuida, se establece una nueva arquitectura de gestión, en el cual, la estación de gestión generará un agente móvil en cuya estructura llevará a la aplicación a cada una de las estaciones para realizar la operación de gestión, y retornar al final con los resultados.

Para la recolección de datos, se hizo uso de PC's conectadas a una red LAN con acceso a Internet y con los softwares requeridos instalados y ejecutados, todo esto se realizó en horas de alto tráfico de información. Los datos obtenidos para ambos enfoques serán mostrados estadísticamente en tablas para luego establecer las semejanzas o diferencias.

1.4 ANTECEDENTES

El mercado de gestión de redes de comunicación sigue siendo dominado a lo largo de décadas por la arquitectura de gestión de Internet, o más usualmente por SNMP. Prácticamente todo el equipamiento de gestión instalado en redes locales están basados en este modelo de gestión, no en tanto este modelo presenta algunas deficiencias importantes, como por ejemplo la falta de escalabilidad, flexibilidad y robustez en redes de dimensión considerable y a la poca eficiencia en el transporte de grandes cantidades de información, y el tráfico que puede ocasionar al realizar todo proceso de gestión.

Estas fallas llevaron a algunas instituciones, empresas privadas y a algunos investigadores tener en consideración estos problemas a los que se deben enfrentar,

y es así que se han establecido diferentes modelos de gestión dentro de las cuales se mencionará y además de algunas investigaciones realizadas, para así tener una idea más amplia del modelo de gestión que se quiere desarrollar.

DISMAN (*Distributed Management-DM*): Promovida por IETF a través de un grupo de trabajo cuyo objetivo es crear una estructura de distribución de carga de gestión por un conjunto de sub-gestores distribuidos.
La gestión distribuida es la delegación de control de una estación de gestión a otra [Bierman98].
Una estación de gestión distribuida, es una estación que recibe solicitudes de otros gestores y ejecutar estas solicitudes realizando las operaciones de gestión en los agentes u otros gestores. Este modelo de distribución permite crear "islas" jerárquicas [LopesOliveira00], aumentando la robustez y la tolerancia a fallas del sistema (Figura1.1). Aunque en situaciones de indisponibilidad de la estación de gestión central hará que los DM lidien localmente ante situaciones críticas.

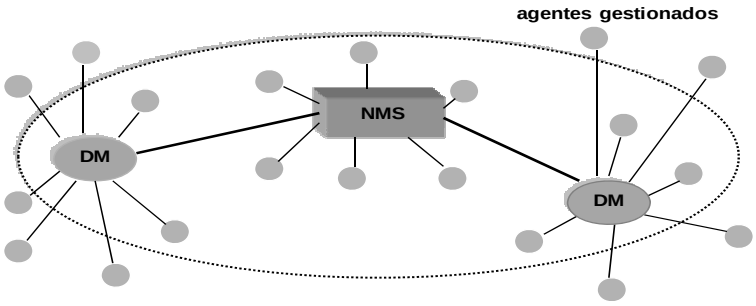


Figura 1.1 Modelo de gestión Disman [LopesOliveira00]

El modelo Disman está basado en los conceptos de *aplicaciones* y de *servicios* de gestión distribuidos. Las *aplicaciones* distribuidas desempeñan funciones de gestión, monitorización y control de los elementos gestionados, y los *servicios* forman el entorno de ejecución para la aplicación de gestión distribuida.
Cada servicio es accedido a través de un modulo MIB (*Management Information Base*) [Bierman98].
Dentro de las deficiencias de este modelo, podemos encontrar que en la recuperación de la información de los diferentes DM es a través de un flujo jerárquico.

WBEM (*Web-Based Enterprise Management*):Es una iniciativa de DMTF (*Distributed Management Task Force*) e incluye un conjunto de tecnologías que permiten la interoperabilidad de gestión de una red empresarial.

DMTF ha desarrollado un núcleo, establecido de estándares que forman el WBEM (Figura 1.2), el cual incluye: un modelo de datos, el estándar CIM *(Common Information Model)*; una especificación de codificación (el CIM-XML); y un mecanismo de transporte (operaciones CIM sobre HTTP) [Sun_Mi06].

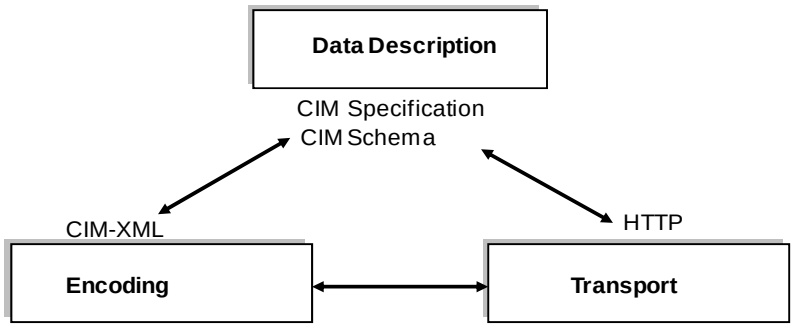


Figura1.2 Relación entre las tecnologías estándar WBEM [Sun_Min06]

Las operaciones CIM sobre HTTP, definirán el plan de operaciones para interoperar de manera abierta, estandarizada y completa con las tecnologías que soporta WBEM, cuya información de gestión convertida a XML será encajonada en un HTTP de carga útil para ser transportado al nodo destino.

En esta arquitectura es necesaria la presencia de un servidor WBEM, y de un cliente WBEM, que se encargará de enviar la solicitud de mensajes.

Dentro de los problemas presentados por esta arquitectura está relacionada a los eventos/alarmas/notificaciones por parte del servidor WBEM, quien necesitaría un HTTP cliente adicional al que ya esta ejecutándose.

CORBA (*Common Object Request Broker Architecture*): CORBA es una de las especificaciones de OMG (*Object Management Group*) para una arquitectura e infraestructura independiente de los vendedores que permiten construir aplicaciones que trabajen sobre redes [OMG]. Un programa desarrollado en CORBA sobre cualquier computador, sistema operativo, lenguaje de programación o red, pueda interoperar con otros programas desarrollado en CORBA o de

cualquier otro vendedor diferente, permitiendo el acceso a la información de forma transparente, sin tener la necesidad de conocer su localización, plataforma o protocolo de transporte (IIOP – *Internet Inter-ORB Protocol*) de los objetos distribuidos (Figura 1.3).

[CORBA] Proporciona una serie de especificaciones de:

1. La sintaxis y semántica del lenguaje de definición de interface (IDL).
2. La interfaz hacia las funciones del ORB.
3. La semántica de pasar un objeto por valor.
4. La Interfaz de Invocación Dinámica (DII) del lado de la interfaz cliente que permite la creación e invocación dinámica de solicitud de objetos.
5. La Interfaz de Esqueletos Dinámicos (DSI) del lado de la interfaz servidor que entrega solicitudes de un ORB para la implementación de un objeto.
6. El repositorio de interfaz que gestiona y proporciona acceso a una colección de objetos.
7. El Adaptador de Objetos Portable (POA), el cual define un grupo de interfaces IDL que se utilizarán para acceder a las funciones ORB.
8. Los mensajes los cuales deben cubrir: la calidad de servicio, y las especificaciones de interoperabilidad en las interfaces de ruteo.

Dentro de las deficiencias de este modelo podemos mencionar:

- No retiene la identidad del host, evitando la localización de los recursos y servicios
- Se establece sobre una arquitectura cliente/servidor que integra la llamada de procedimiento remoto (RPC), no asegurando su flexibilidad y reconfigurabilidad.

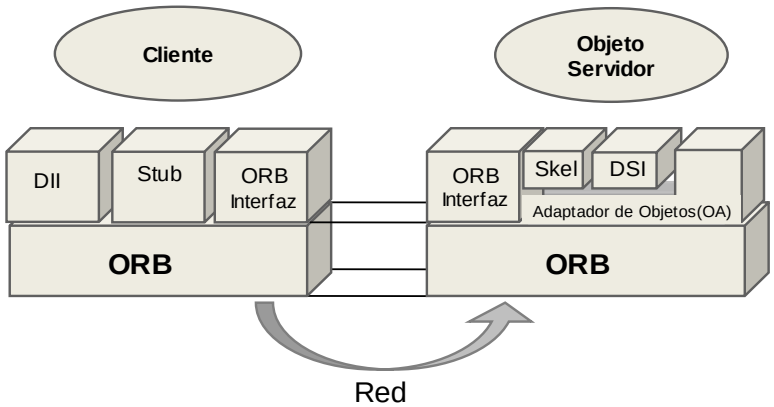


Figura 1.3 Arquitectura de comunicaciones en CORBA [García]

1.5 TRABAJOS RELACIONADOS

Dentro de las investigaciones desarrolladas por otros investigadores podemos mencionar a:

a. ADAPTIVE MANAGEMENT OF EMERGING BATTLEFIELD NETWORK: Tesis desarrollada por Dimitrios Fountoukidis [Fountoukidis04], que propone una arquitectura de gestión red de computadores adaptable mediante una capa de Inteligencia Artificial (IA) que contiene un número necesario de agentes con IA que serán alimentados con los datos de la capa de agentes móviles y sustituir así, a la capa de aplicación. Su arquitectura toma como base el proyecto MANTRIP *(MANagement, Testing and Reconfiguration of IP)*, basadas en redes IP [Mota].

b. GESTÃO DISTRIBUIDA EM SNMP: Tesis de investigación desarrollada por Rui Sánchez de Castro Lopes [Rui02] quien hace un estudio detallado de las tecnologías de distribución de gestión en ambiente SNMP, para después elaborar un trabajo de distribución de operaciones, como por ejemplo las herramientas definidas por el grupo de trabajo DISMAN y luego establecer una integración con la plataforma de agentes móviles.

Dicha asociación le permitirá definir una arquitectura de gestión el cual cubrirá todos los grados de distribución, desde el método centralizado pasando por la distribución débil y fuerte, permitiendo al mismo tiempo una gestión cooperativa.

Su resultado debe ser visto como un middleware de herramientas estándar y abierta que permitan diseñar arquitecturas de gestión distribuidas de acuerdo con las políticas de gestión definidas por el desarrollador.

c. MANAGING AGENT PLATFORM WITH THE SIMPLE NETWORK MANAGEMENT PROTOCOL: Tesis de investigación desarrollada por Brian Douglas Remick [Remick02], quien desarrolla un esquema de gestión de red centralizada denominada AgentSNMP, cuya meta es proporcionar un método formal de gestión de plataforma de agentes usando el paradigma de gestión de red. Encuentra similaridades entre la red y la plataforma de agente, para luego aplicar un esquema de gestión de red centralizada para la gestión de plataformas de agentes. En este caso, los recursos gestionables son la

plataforma de agente misma (y todos los host que la abarcan), y un agente(s) proxy que proporcionen una vista estandarizada de la plataforma de agente al sistema de gestión. Utiliza la plataforma JADE, debido a sus especificaciones en términos de agentes con las especificaciones FIPA, proporcionándole una solidez de plataforma.

Habiendo ya conocido algunos sistemas ya establecidos, así como de algunas investigaciones desarrolladas, todas direccionadas a la gestión de red, y que de una manera u otra tratan de lidiar con el tráfico ocasionado en la red. Se propone un nuevo modelo de sistema de gestión mediante la aplicación de un agente móvil cuyas características: reducción de tráfico, independiente del sistema operativo, y tolerantes a fallas nos permitirán obtener información confiable y veras en todo proceso de gestión.

CAPITULO 2 Gestión de Red

2.1 INTRODUCCIÓN

Debido al constante crecimiento de las tecnologías en las redes, en las cuales tienden a ser cada vez más largas y complejas, y además soportar dispositivos de los diferentes tipos de vendedores, entonces gestionar una red es ya un problema difícil y tedioso.

Esto conllevará a involucrar herramientas de gestión y protocolos que soporten a los diferentes dispositivos propietarios.

La gestión de una red la podemos definir desde varios puntos de vistas:

- [Hegering]: Es la gestión de sistemas interconectados comprendiendo todas las medidas necesarias para asegurar una operación efectiva y eficiente del sistema y sus recursos según los objetivos de la organización. El propósito de la gestión es de disponer de los servicios y aplicaciones de un sistema interconectado con el nivel deseado de calidad y garantizar la disponibilidad y un rápido, despliegue flexible de los recursos interconectados.

  Si la prioridad es la gestión de red de comunicaciones y sus componentes, se hablará de gestión de red; si el énfasis está en los sistemas finales, se referenciará a la gestión de sistemas. La aplicación de gestión es la responsable de que la aplicación y servicios estén disponibles en el sistema distribuido. Como vemos la importancia está en que, el ámbito de la

gestión no se restringe a la red, sino a los sistemas que se interconecta y las aplicaciones y servicios que proporcionen dichos sistemas.

- [Boutaba]: La gestión de red es la gestión de recursos y servicios, el cual incluye: control, monitoreo, configuración, actualización y reporte del estado del dispositivo y de los servicios de red, simplificando las complejidades de gestión de red, que es la tarea de todo sistema de gestión, permitiendo así extrapolar la información de gestión dentro de la forma que sea gestionable por el ser humano. Los servicios confiables deben proporcionar a la red alta Calidad de Servicio (QoS) minimizando las caídas de la red, detectando y solucionando estas caídas y sus errores. Y por último el consciente costo de toda gestión es tener por finalidad guardar los rastros de los recursos y de los usuarios de red, los cuales deben ser rastreados y reportados.
- [ANSI.T1]: La gestión de red consiste en la ejecución de un conjunto de funciones requeridas para controlar, planear, asignar, desplegar, coordinar y monitorizar los recursos de una red de Telecomunicaciones, incluyendo funciones de rendimiento tales como: planear la red inicial, asignación de frecuencias, encaminamiento de tráfico predeterminado para soportar balance de carga, autorización de la distribución de claves criptográficas, gestión de configuración, gestión de fallas, gestión de seguridad, gestión de rendimiento y gestión de contabilidad, como vemos, se hace bastante hincapié en las áreas funcionales de gestión de red, siguiendo el modelo FCAPS (Fault, Configuration, Accounting, Performance y Security).

Por otro lado hay que hacer una interesante observación, que en el sector IP los datos de gestión de red es siempre tratada como “ciudadanos de segunda clase” comparado con los datos de usuario [Boutaba], puede llegar a ser verdadero, ya que el transporte de datos de gestión nunca entran en el camino del transporte de datos del usuario, lo importante es que los datos de gestión está en constante ascenso, especialmente en la demanda de los servicios de Calidad de Servicio (QoS) en tiempo real, y más aún, ahora que está utilizando un protocolo de

transporte orientado a conexión como TCP para HTTP, que implica llevar datos de gestión al mismo nivel de los datos del usuario, a través de los routers de red.

2.2 STATUS Y FUTURO DE LA GESTIÓN DE RED

La meta de toda gestión de red es asegurar que los usuarios de una red reciban los servicios de tecnologías de información con la calidad de servicio que esperan. Por lo tanto el administrador de red necesita una herramienta de gestión para que pueda monitorear la disponibilidad de la red, así como su utilidad y performance dentro de la industria y que sean aceptables a los estándares como sea posible para reducir los conflictos en los sistemas de gestión, y establecer las acciones correctivas.

El propósito de todo monitoreo de red es de recoger información acerca del status y comportamiento de los elementos de la red, el cual incluirá información estática, relacionada a la configuración, información dinámica, relacionada a eventos en la red, e información estadística, resumidamente de la información dinámica [dit00].

Típicamente el dispositivo gestionado en la red incluye un módulo de agente responsable para la recolección de información y transmitirla a uno o más estaciones de gestión. Cada estación de gestión incluye software de aplicación de gestión de red además del software para comunicarse con los agentes (Figura 2.1) La información puede se activamente recogida por intermedio de polling (sondeo) por la estación de gestión, o pasivamente por intermedio de reporte de eventos por el agente.

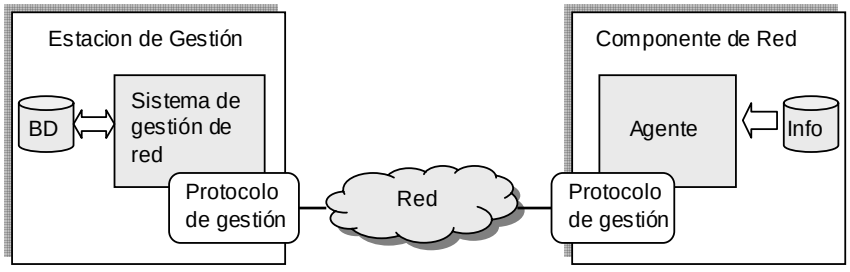


Figura 2.1 Modelo simplificado de un sistema de gestión de red [ELLS].

Toda estación de gestión de red (NMS) presenta mecanismos de generación de comandos y de recepción de notificaciones. El intercambio de notificaciones y de

comandos entre entidades de gestión es efectuada por un protocolo especifico, el protocolo de gestión.

Por otro lado, el intercambio de información depende de varios parámetros de funcionamiento relativos a la entidad específica, dentro de un conjunto de entidades o a la propia infraestructura de comunicación.

El IAB (*Internet Architecture Board*) o Comité de Arquitectura de Internet ha elaborado o adoptado varias normas para la gestión de red. En su mayoría, éstas se han diseñado específicamente para ajustarse a los requisitos de TCP/IP, aunque cuando sea lo posible deben de cumplir con la arquitectura OSI, dentro de las cuales podemos mencionar a:

2.2.1 GESTIÓN OSI

Conjunto de estándares desarrollados conjuntamente por ISO y CCITT para la gestión de redes OSI (*Open Systems Interconnection*). Es el más complejo y el más amplio de todos los desarrollados por ISO.

CCITT tiene reservada de la serie de números, el número X.700 [X.700] para este conjunto de estándares e incluye la definición de:

- Un servicio de gestión (CMIS - *Common Management Information Service*).
- Un protocolo de gestion (CMIP - *Common Management Information Protocol).*
- Una base de datos.

En OSI se utiliza el término "**gestión de sistemas**" en vez de gestión de red [GestionOSI]. Las directivas de gestión fueron establecidas por ISO 7498-4 [X.700], que especifica el entorno de gestión para el modelo OSI, y luego complementada por ISO 10040 [X.701].

Un equipo gestionado dentro del entorno OSI seguirá el siguiente modelo de arquitectura:

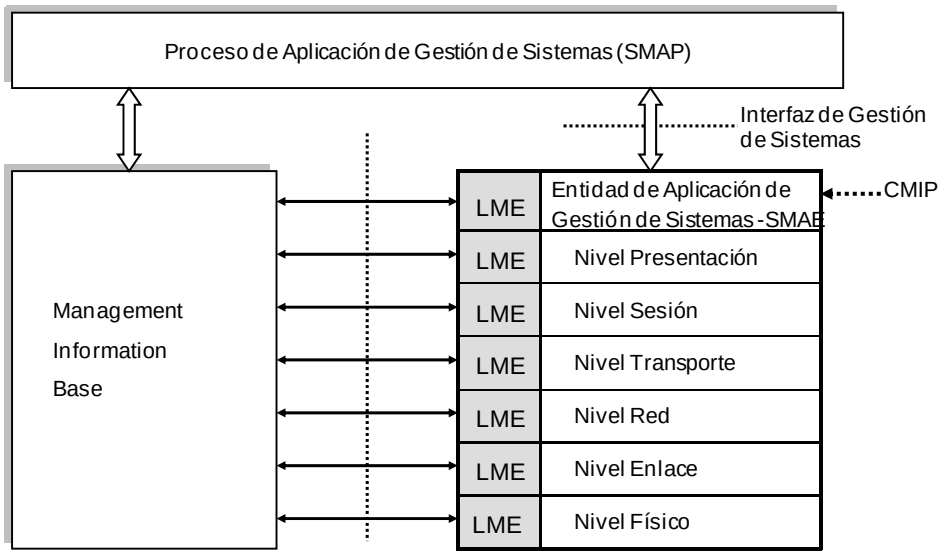


Figura 2.2 Modelo de arquitectura CMIS [GestionOSI]

Los elementos clave de este modelo de arquitectura son:

- El Proceso de Aplicación de Gestión de Sistemas (SMAP - S*ystems Management Application Process*), que es el software local de un equipo (sistema) gestionado. Implementa funciones de gestión y coordina con otros SMAPs de otros sistemas.
- La Entidad de Aplicación de Gestión de Sistemas (SMAE - S*ystems Management Application Entity*) que es la entidad de nivel de aplicación, responsable del intercambio de información de gestión con SMAEs de otros nodos, especialmente con el sistema que hace las funciones de centro de control de red. Para esta función se utiliza un protocolo normalizado (CMIP).
- La Entidad de Gestión de Nivel (LME - L*ayer-Management Entity*) que proporciona funciones de gestión específicas de cada capa del modelo OSI.
- La Base de Información de Gestión (MIB).

El intercambio de información entre dos entidades (gestor, agente) es una de las funciones básicas del sistema de gestión OSI. Esta función de intercambio de información en el sistema de gestión OSI se conoce como CMISE (C*ommon Management Information Service Element*), que se divide en dos partes: la interfaz con el usuario que especifica los servicios proporcionados, denominado CMIS para la realización de las operaciones de gestión mediante primitivas de servicios y

el protocolo, que especifica el formato de las PDUs (*Packet Data Unit*), denominado CMIP.

ISO también estableció una serie de formalismos, conocidos como GDMO (*Guidelines for Definition of Management Objects*) [GDMO] con la finalidad de especificar los objetos de gestión. Este consiste en un lenguaje de especificación de clases de objetos, que especifican su comportamiento, atributos y herencia, en donde los objetos de gestión son representaciones lógicas de entidades físicas o recursos asociados a un sistema o subsistema.

ISO ha dividido la gestión de red en cinco áreas funcionales [X.700], dentro de las cuales quizás no pueda existir una funcionalidad común ya que dependerá del tipo de red gestionada, del tipo de equipos gestionados y de los objetivos de la gestión de red [dit00], en la cual podemos detallarlos de la siguiente manera:

- Gestión de Fallas: El objetivo es identificar las fallas tan rápido como sea posible después de que ocurran, e identificar las causas que lo originaron, así como la acción de diagnostico a tomar. Comprensivamente la gestión de fallas es la más importante tarea en toda gestión de red.

  Esta herramienta de gestión puede ayudar al incremento de confiabilidad de la red para una identificación rápida de la falla, aislando la causa, si es posible, su corrección.

- Gestión de Configuración: La gestión de configuración trata con la inicialización, modificación, y shutdown (suspensión) de la red. Las redes están continuamente adaptándose a los nuevos dispositivos en la que son agregados, removidos, reconfigurados, o actualizados. El proceso de gestión de configuración involucra la identificación de los componentes de la red y sus conexiones, recogiendo de cada dispositivo la información de configuración, así como la relación entre sus componentes. Para ejecutar estas tareas, el gestor de red necesita la información topológica, y la información de los dispositivos que componen la red.

- Gestión de Performance: La gestión de performance involucra la medida de performance de una red y los recursos en términos de utilización, throughput (rendimiento), tasa de error, y tiempo de respuesta. Con la información de gestión de performance, un gestor de red puede reducir o

prevenir la sobrecarga de la red y la inaccesibilidad. Este parámetro ayudará a proporcionar un nivel de consistencia al usuario sobre la red, sin someter a esfuerzo excesivo la capacidad de los dispositivos y enlaces.

- Gestión de Contabilidad: En el área de Gestión de contabilidad, la monitorización está interesado en la recolección de información utilizada para la adecuada contabilidad. Este tipo de información ayuda a un gestor de red asignar de manera correcta los recursos, así como, el plan de crecimiento de la red.

- Gestión de Seguridad: Esta gestión establecerá funciones que proporcionen protección continua a la red y sus componentes en sus distintos aspectos, como por ejemplo el acceso a la red, al sistema y a la información en tránsito a través de políticas de seguridad.

Podemos concluir que toda gestión de red consiste en la planificación, organización, supervisión y control de los elementos de comunicación para garantizar un buen servicio y de acuerdo a un coste.

2.2.2 GESTIÓN SNMP

Comprende la utilización del protocolo simple para la gestión de red.

2.2.2.1 SNMP - Protocolo Simple de Gestión de Red

La arquitectura de gestión basada en SNMP surgió en los fines de la década de los 80 como una solución provisoria coaccionada esencialmente para sistema TCP/IP. Esta ha sobrevivido hasta estos días pasando por tres grandes evoluciones (Figura 2.3).

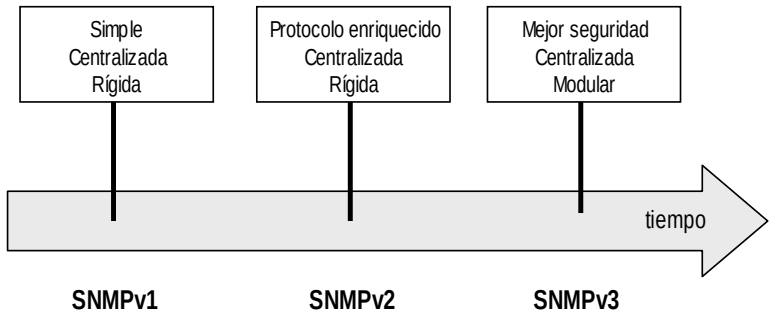


Figura 2.3 Evolución de la arquitectura SNMP [Rui02].

La primera versión de la arquitectura SNMP se encuentra descritas en apenas tres documentos lo que posibilita una rápida comprensión y facilita el desenvolvimiento de aplicaciones de gestión [RFC1155] [RFC1156] [RFC1157]. Esta simplicidad despertó la atención del mercado que de prisa la adaptó para sus productos. Actualmente, aún es la versión más común en todo el mundo.

Una versión mejorada de SNMP, llamada SNMPv2 fue puesta en circulación en 1993 y revisada en 1995. El último estándar llamado SNMPv3, editada en 1998, define un esquema completo para las versiones presentes y futuras por su característica de seguridad añadida a SNMP.

Cada equipo conectado a la red ejecuta unos procesos (agentes), para que se pueda realizar una administración tanto remota como local de la red. Dichos procesos van actualizando *variables* (especie de históricos) en una base de datos, que máquinas remotas pueden consultar, por ejemplo, en el caso de un router; los interfaces activos, la velocidad de sus enlaces, bytes emitidos, bytes recibidos, en una impresora; que se terminó el papel, en un MODEM; la pérdida de conexión, y en un switch; puertos conectados, etc.

Las especificaciones necesarias para todo sistema de gestión son:

- RFC 1157: Define al protocolo SNMP y a su arquitectura aplicada para la gestión del MIB y de su contenido.
- RFC 1155: SMI (*Structure of Management Information*) o Estructura de la Información de Gestión para redes TCP/IP, que describe como está definida el MIB.
- RFC 1213: MIB-II (*Management Information Base)* o Base de Información de Gestión para redes TCP/IP, que describe el contenido de dicha MIB.

SNMP es un protocolo de nivel aplicación para consulta a los diferentes elementos que conforman una red, y encaminado a operar sobre el Protocolo UDP *(User Datagram Protocol*). La estación de gestión comunicará la información usando SNMP, el cual se encuentra implementado encima de los protocolos de enlace de datos, IP y UDP. (Figura 2.4).

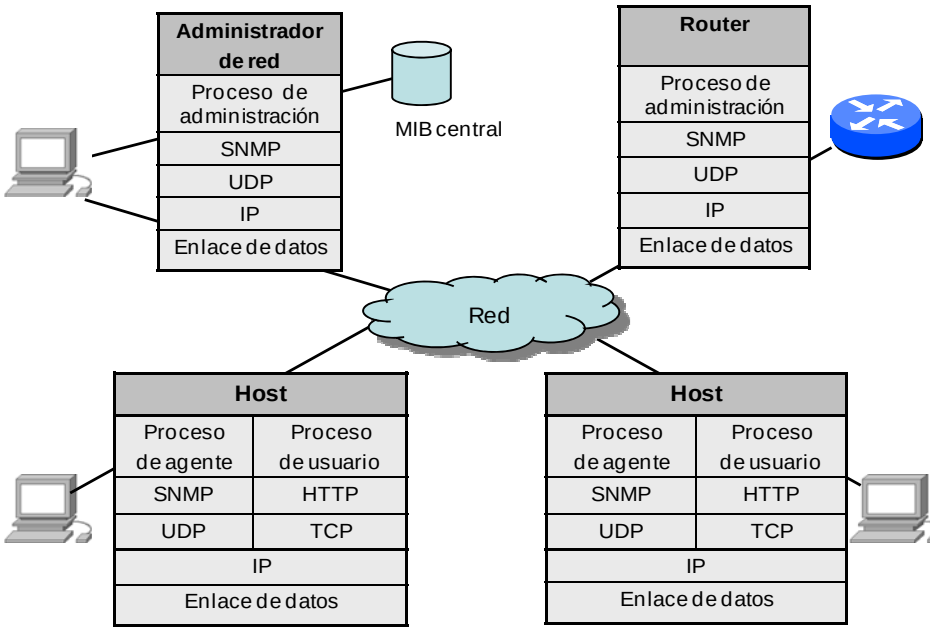


Figura 2.4 Componentes de la arquitectura SNMP [ELLS].

SNMP facilita la comunicación entre la estación de gestión y el agente de un dispositivo de red (nodo gestionado), permitiendo que los agentes transmitan datos estadísticos (*variables*) a través de la red a la estación de gestión. En suma SNMP separa la arquitectura de gestión y la de la arquitectura de hardware del dispositivo.

2.2.2.2 ARQUITECTURA SNMP

Hay cuatro elementos principales en la gestión de red SNMP:

A. Estación de Gestión de Red: Típicamente es un dispositivo stand-alone, que proporciona la interfaz para que el administrador de red pueda interactuar con el sistema de gestión. La estación de gestión tiene la capacidad de configurar, monitorizar, analizar y controlar los diferentes componentes que comprende la red. Contiene el software de gestión, y mantiene una base de datos denominada MIB con formato SMI.

B. El Agente de Gestión: Es un programa de software instalado en el dispositivo, que responde a las solicitudes emitidas por la estación de gestión. El agente también puede enviar a la estación de gestión información no solicitada, conocida como "trap" o interrupción.

Actualmente todos los dispositivos en una red tienen un agente de gestión y debe de ser instalado, en caso contrario, o alternativamente establecer un agente "proxy".

Un agente proxy es el programa que soporta dispositivos sin disponibilidad de implementación SNMP (muchos dispositivos móviles en el mercado no tienen implementación SNMP). El proxy es un gestor SNMP que solicita servicios desde la consola de gestión, de parte de uno o de un número de dispositivos no disponibles SNMP.

C. Base de Información de Gestión: Es la agrupación de objetos o valores de datos representando el aspecto del dispositivo gestionado, permitiendo así el intercambio de información entre el gestor y el agente. Los objetos gestionados están organizados dentro de un árbol de jerarquías cuya estructura es la base para el esquema de nombramiento SNMP [O'Reilly01], al cual se le puede adicionar y ser completado fácilmente, y obtener información específica de manera transversal de dicho árbol.

La Estructura de Información de Gestión, el cual está referenciada en [RFC1155], establece que cada objeto gestionado debe tener los siguientes elementos: un nombre, una sintaxis y una codificación. Actualmente se está estudiando una nueva versión de SMI, el SMIng (SMI *next generation*) el cual permitirá definir más claramente el lenguaje de los datos [RFC3780].

Cada objeto es inequívocamente identificado por un nombre OID (*Object Identifier*), que consiste en una secuencia de números enteros separados por puntos ("."), por el cual cada número identifica un nodo dentro del árbol de jerarquías, que a su vez puede estar ligado a otros nodos, también numerados, constituyendo así un subárbol. El proceso puede continuar hasta un número indeterminado de clases. Por ejemplo, la secuencia de nuestra raíz -1-3-6-1-2-1 designa el identificador .1.3.6.1.2.1, o sea, .iso.org.dod.internet.mgmt.mib-2 (Figura 2.5).

La sintaxis de un determinado tipo de objeto definirán la estructura de datos del objeto, y puede ser cualquiera de los tipo básicos definidos por

ASN.1 (*Abstract Syntax Notation One*), como INTEGER, OCTET STRING, OBJECT IDENTIFIER, SEQUENCE, SEQUENCE OF, o de tipos definidos a nivel de aplicación, como Integer32, IpAddress, Counter32, Gauge32, Unsigned32, Timetick, Opaque, o Counter64, y de alguna convención textual (por ejemplo, DateAndTime, DisplayString, PhysAddress).

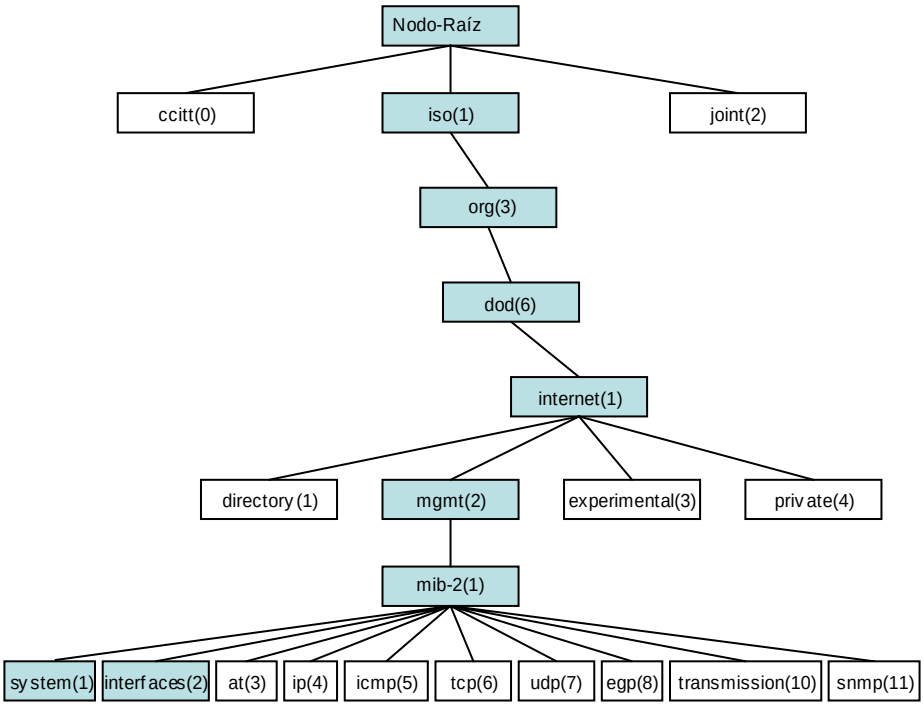


Figura 2.5 Árbol de identificación de objetos [O'Reilly01].

SNMP utiliza la codificación BER (*Basic Encoding Rules*) [Steffen02], que describe como la información está asociada con los objetos gestionados.

La popularidad de SNMP ha dado resultados en el desarrollo de estándares de almacenamiento de datos críticos de operación de red, donde el MIB-II almacena datos sobre tráfico TCP/IP, enrutamiento, configuración, y errores, además de tener implementado soporte para dispositivos multi-protocolos que permiten a los sistemas de gestión de red controlar la operación SNMP.

D. Protocolo de Gestión de Red: Es el último elemento del modelo de gestión de red, que enlaza la estación y al agente para especificar las reglas

de comunicación. El protocolo utilizado para la gestión de redes TCP/IP es el SNMP.

SNMP define ocho mensajes que pueden enviarse:

GET REQUEST: Solicita uno o más atributos (valores) de un objeto o variable, por ejemplo, la cantidad de memoria disponible en la CPU. Transmitido por el nodo administrador y recibido por el agente que contesta.

GET-NEXT REQUEST: Es un tipo extendido de solicitud de mensaje en el cual se solicita el siguiente atributo de un objeto. La solicitud GET-NEXT es usado bastante en tablas dinámicas, así como en tablas de ruteo interno IP. Transmitido por el nodo administrador y recibido por el agente que contesta.

GET-BULK REQUEST: Solicita un conjunto amplio de atributos en vez de solicitar uno por uno, minimizando el número de protocolos intercambiables requeridos para recuperar una cantidad larga de información de gestión. Transmitido por el nodo administrador y recibido por el agente.

SET REQUEST: Es un mensaje que sirve para actualizar uno o varios atributos de un objeto. Transmitido por el nodo administrador y recibido por el agente.

INFORM REQUEST: Describe la base local de información de gestión MIB para intercambiar información entre los nodos de administración. Es transmitido por el nodo administrador y recibido por otro nodo administrador.

GET RESPONSE: Devuelve los atributos solicitados. Transmitido por el agente y recibido por el nodo administrador.

SET NEXT REQUEST: Actualiza el siguiente atributo de un objeto. Transmitido por el nodo administrador y recibido por el agente.

NOTIFY: También llamado mensaje trap. Es un mensaje no solicitado que es enviado por un agente a un sistema de gestión cuando el agente detecta

un tipo de evento inevitable, o la pérdida de comunicación con el vecino. Transmitido por el agente y recibido por el nodo administrador.

La siguiente figura 2.6 muestra los diferentes tipos de mensajes que pueden enviarse mediante el protocolo SNMP [GestionOSI].

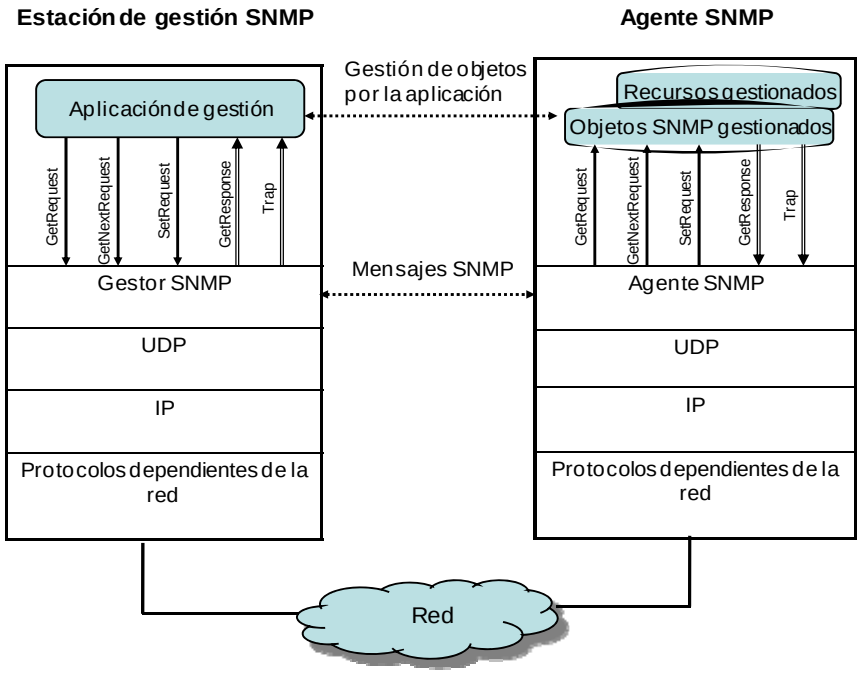


Figura 2.6 Tipos de mensajes SNMP [GestionOSI].

SNMP es independiente del protocolo, y se puede implementar utilizando comunicaciones UDP o TCP, pero por norma general se suelen usar comunicaciones UDP y está orientada a la gestión de equipos y servicios sobre redes de datos.

2.2.3 GESTIÓN TMN

El término TMN (*Telecommunication Management Network*) fue introducido por ITU-T (antiguo CCITT) como una abreviación para la 'Gestión de Redes de Telecomunicaciones'. El concepto de TMN está definido por las recomendaciones M.3010. TMN tiene una fuerte relación con la gestión OSI y define un número de conceptos que tienen una relevancia para la Gestión de Internet [Pras99].

De acuerdo a M.3010. "Un TMN es conceptualmente una red separada que hace de interfaz a los diferentes puntos o nodos" en una red de telecomunicaciones.

La figura 2.7 muestra la relación entre un TMN y la red de telecomunicaciones gestionada.

El punto de interfaz entre el TMN y la red de telecomunicaciones están formadas por el *sistemas de transmisión* e *intercambios* conectados vía una red de comunicaciones de datos a una o más sistemas de operaciones, las cuales ejecutan funciones de gestión; estas funciones pueden ser llevadas externamente por operadores humanos pero también automáticamente. Es posible que una simple función de gestión sea ejecutada por múltiples sistemas de operaciones. En este caso la red de comunicación de datos es utilizada para el intercambio de información de gestión entre los sistemas de operaciones. La red de comunicación de datos es también utilizada para conectar workstations, el cual permitirá al operador interpretar información de gestión.

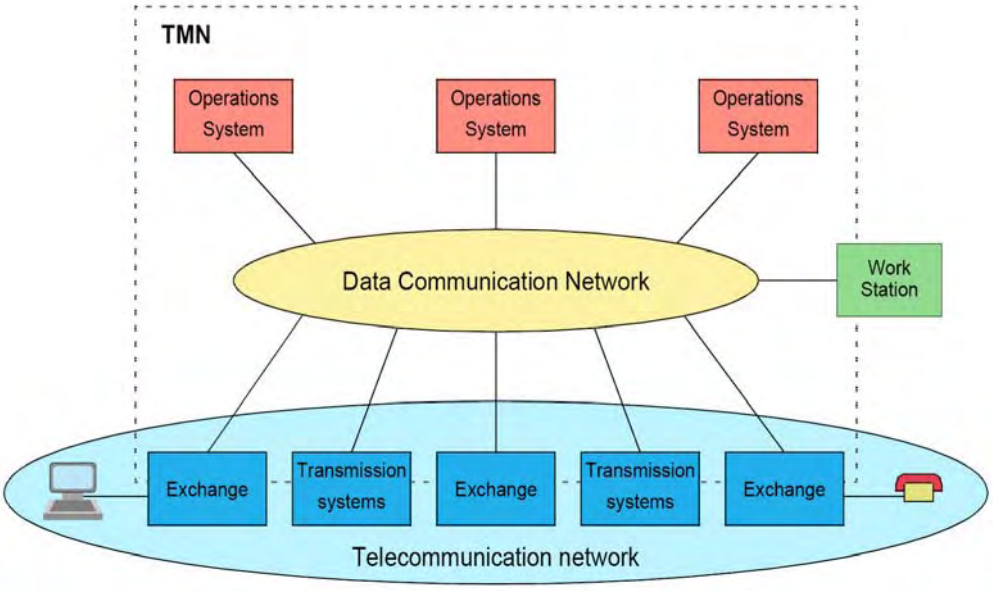


Figura 2.7 Relación general de un TMN en una red de Telecomunicaciones [Pras99].

En las recomendaciones M.3010 se definen los conceptos de gestión TMN e introduce las diferentes arquitecturas de gestión en sus diferentes niveles de abstracción como:

- Una arquitectura funcional, el cual describe un número de funciones de gestión.
- Una arquitectura física, el cual describe como estas funciones de gestión pueden ser implementadas dentro de los equipos físicos.

- Una arquitectura de Información, que describe los conceptos que han sido adoptados desde la gestión OSI.
- Una arquitectura de capas - LLA (*Logical Layered Arquitecture*), el cual incluye una de las mejores ideas de TMN en un modelo que muestra como la gestión puede ser estructurada de acuerdo a las diferentes responsabilidades.

Los sistemas de gestión TMN están concebidos por ser capaces de gestionar equipos y redes de telecomunicaciones como: redes telefónicas, ISDN, y redes de servicios móviles, haciendo uso de sus propios protocolos, estructura y definición de su información.

CAPITULO 3 SNMP en Plataforma Windows y Arquitectura AdventNetSNMP

3.1 INTRODUCCIÓN

En este capítulo se describirá al S.O (Sistema Operativo) Windows de Microsoft, desde el punto de vista de gestión de red, ya que éste maneja SNMP dentro de su plataforma, aparte de proporcionar y soportar un agente basado en el RFC-1213 MIB. Este S.O fue utilizado como la plataforma base para el desarrollo de la aplicación, el cual fué instalado tanto en la estación central como en las estaciones a gestionar.

Para el desarrollo de la arquitectura de gestión se describirá al AdventNet SNMP API, teniendo en cuenta el acceso a su información y de su contenido, hay que mencionar que es un software comercial, la cual se encuentra disponible en AdventNet Inc, pero también se encuentra disponible para su evaluación por un periodo de tiempo (45 días), y éste es el caso que se utilizará para el procedimiento experimental. Para el desarrollo de este modelo se utilizó los APIs de alto-nivel, ya que los paquetes en este nivel tienen inteligencia SNMP [AdventNet04]. Esta arquitectura, está totalmente desarrollado en JAVA, por el cual se puede tomar ventaja de su alta portabilidad, proporcionando un lenguaje de programación multiplataforma [Schildt05].

Ambos sistemas serán explicados de manera entendible y detallada en este capítulo. Pero antes describiremos la implementación del protocolo SNMP dentro

del Sistema Operativo Windows Server 2003 de la familia de Microsoft y a la arquitectura AdventNet SNMP API 4.

3.2 IMPLEMENTACIÓN DEL PROTOCOLO SNMP DE WINDOWS MICROSOFT

La implementación del SNMP de Windows es un servicio de 32 bits que soportan las computadoras que están ejecutando los protocolos TCP e IPX. Es un servicio opcional de Windows de Microsoft y puede ser instalada después de que TCP/IP e IPX hayan sido configuradas con éxito. Windows implementa las versiones SNMPv1 y v2c. Estas versiones están basadas en los estándares industriales que definen como la información de gestión de red está estructurada, almacenada, y comunicada entre los sistemas de gestión para redes basadas en TCP/IP [Fountoukidis04].

El servicio SNMP de Windows proporciona un agente que permiten centralizar, gestionar remotamente a las computadoras que estén ejecutándose en este sistema operativo, donde la estación de gestión viene a ser el cliente y el servidor cada una de las entidades gestionadas.

Para usar la información que proporciona el servicio SNMP de Windows, debemos tener al menos un host central local que este ejecutando una aplicación de software de gestión SNMP. El servicio SNMP de Windows sólo proporciona el agente SNMP, no incluyendo el software de gestión SNMP.

Windows Server cuenta con dos interfaces de programación de aplicación (APIs) en la cual podemos desarrollar nuestro propio software de aplicación de gestión SNMP.

- API WinSNMP (Wsnmp32.dll), el cual proporciona un conjunto de funciones como codificación y decodificación de mensajes.
- API de Gestión (Mgmtapi.dll), el cual proporciona las funciones básicas para el desarrollo de un sistema de gestión.

El SNMP de Windows soporta además los servicios de: MIB II de Internet, MIB II de Gestor LAN, MIB de Recursos de Host y MIB’s propietarios de Microsoft.

3.2.1 ARQUITECTURA SNMP DE WINDOWS

La arquitectura interna de implementación de SNMP en Windows Server[1] está dividida dentro de las funciones de gestor y agentes, en algunos casos estas funciones se sobreponen, como ilustra la figura 3.1.

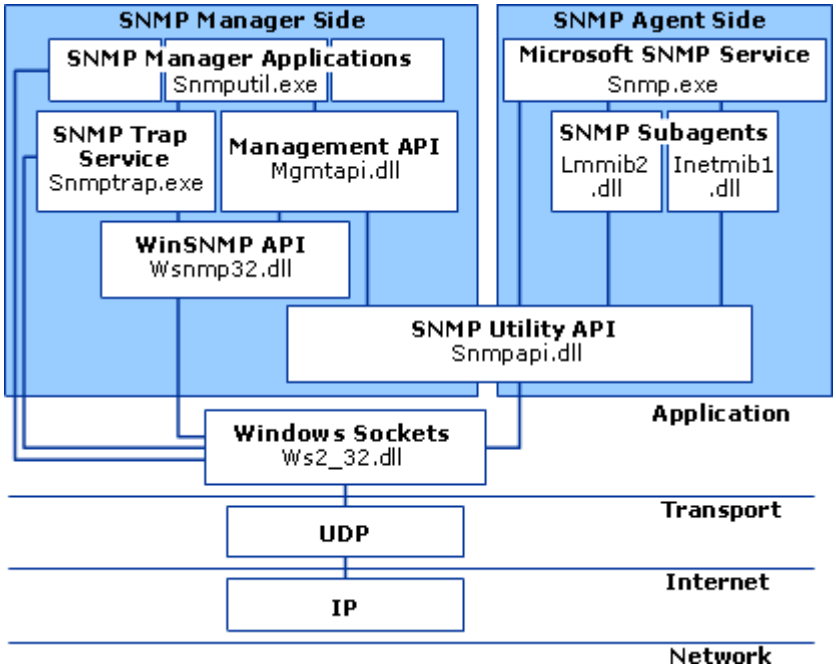


Figura 3.1. Arquitectura SNMP de Windows Server 2003 [SNMPWorks].

Por defecto, SNMP hace uso del protocolo UDP que utiliza el puerto 161 para atender los mensajes polling y el puerto 162 para atender los mensajes traps. Se pueden cambiar estos puertos colocando y configurando el archivo local de servicios.

3.2.1.1 FUNCIONAMIENTO DE SNMP

El ejemplo ilustrado en la figura 3.2 muestra como se comunica los sistemas de gestión con el agente.

[1] http://technet.microsoft.com/en-us/library/cc783142.aspx

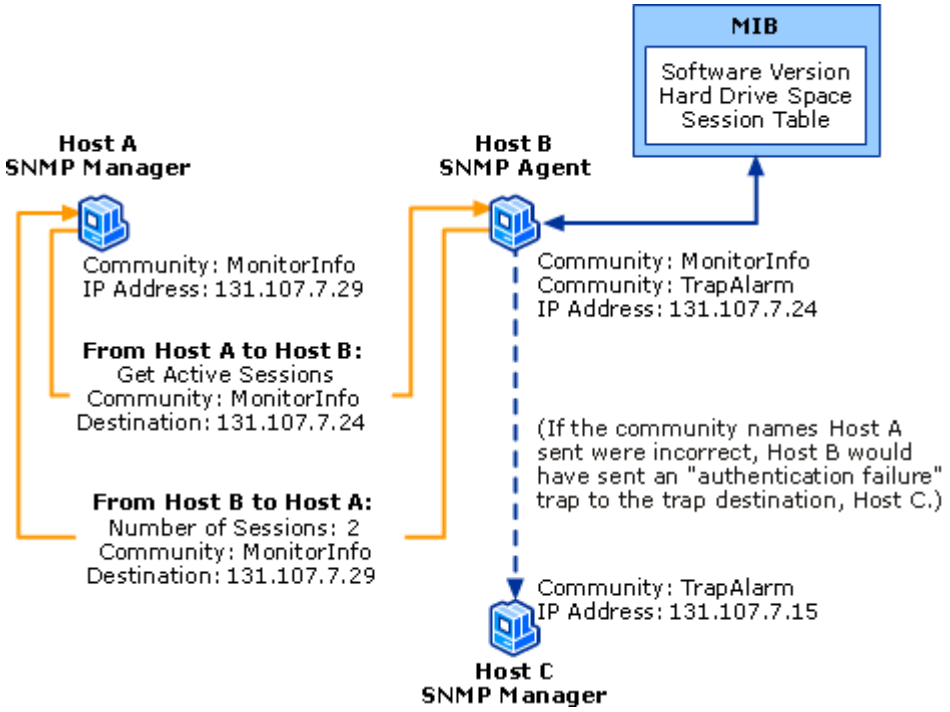


Figura 3.2. Interacción entre el gestor y el agente SNMP [SNMPWorks]

El proceso de comunicación es como sigue:

- El gestor SNMP, el Host A, forma un mensaje SNMP que contiene una información de solicitud (Get) por el número de sesiones activas, el nombre de la comunidad al cual el gestor SNMP pertenece, y el destino del mensaje – la dirección IP (131.107.7.24) del agente SNMP, el Host B.

  El gestor SNMP puede utilizar la librería API de Gestión SNMP de Microsoft (Mgmtapi.dll) o la librería API WinSNMP de Microsoft (Wsnmp32.dll) para ejecutar este paso.

- El gestor envía la información de solicitud al Host B utilizando la librería de servicio SNMP.

- Cuando el Host B recibe el mensaje, se verifica que el nombre de la comunidad (MonitorInfo) contenida en el paquete esté en la lista de nombre de comunidad aceptable y la lista de permisos de acceso por la comunidad, luego se verifica la dirección IP fuente.

- Si el nombre de la comunidad o el permiso de acceso es incorrecto, y el servicio SNMP ha sido configurado para enviar un trap de autentificación,

el agente enviará un trap “authentication failure” al destino del trap especificado, el Host C. El Host B, y el Host C deben pertenecer a la misma comunidad: TrapAlarm.

- El agente ‘master’ componente del agente SNMP llama a la extensión del agente para recuperar la información de sesión solicitada desde el MIB.
- Utilizando la información de sesión recuperada desde la extensión de agente, el servicio SNMP forma un mensaje SNMP de retorno que contiene el número de sesiones activas y el destino – la dirección IP (131.107.7.29) del gestor SNMP, el Host A.
- El Host B envía la respuesta al Host A.

3.2.1.2 COMUNIDAD

Cada host de gestión y el agente SNMP deben pertenecer a la misma comunidad SNMP. Una comunidad SNMP es una agrupación de host congregados para propósitos de administración, entonces es necesario decidir que computadoras deben pertenecer a la misma comunidad, el cual no siempre está determinada por la proximidad física de las computadoras. Las comunidades son identificadas por nombres únicos asignados a ellos.

Los nombres de comunidad son utilizadas para autentificar los mensajes SNMP y así proporcionar una esquema de seguridad rudimentaria para el servicio SNMP. Aunque un host puede pertenecer a varias comunidades al mismo tiempo, pero un agente SNMP no puede aceptar solicitudes de un sistema de gestión de una comunidad que no se encuentre en la lista de nombres de comunidades. No hay una relación entre los nombres de comunidad y nombres de dominio o nombres de grupo de trabajo. Un nombre de comunidad puede ser considerada como un password oculto por la consola de gestión SNMP y por las computadoras gestionadas. Nuestra responsabilidad como administrador es colocar nombres de comunidades difíciles de adivinar cuando instalemos el servicio SNMP.

3.3 ADVENTNET SNMP API

AdventNet SNMP API es un amplio kit de herramientas para el desarrollo rápido de aplicaciones de gestión basado en SNMP que son confiables, escalable e independiente del sistema operativo [AdventNet04].

Los desarrolladores de gestión de red pueden hacer uso de sus librerías para construir: aplicaciones de gestión, aplicaciones standalone y aplicaciones basados en Web, y además de los componentes de software encajonados y distribuidos por EJB *(Enterprise Java Bean)*, CORBA, y aplicaciones RMI *(Remote Method Invocation).* Las librerías proporcionan diferentes funciones y componentes comúnmente utilizadas fuera de su recinto para hacer que el desarrollo sea más simple.

El núcleo de AdventNet es un conjunto de APIs hechos en Java, y por lo tanto puede estar integrada en cualquier aplicación típica de Java. En suma, los APIs proporcionan las interfaces de despliegue que lo habilitarán dentro de entornos distribuidos a través de RMI, CORBA, y contenedores J2EE, y construir aplicaciones en tiempo real que nos permitan optimizar el performance, monitorización y la localización de los elementos de red.

3.3.1 BENEFICIOS

- Soporta todas las plataformas, como: Solaris, Windows NT, y Linux con un código base común.
- Es escalable, fundamentalmente diseñada como un sistema multi-capa, basada ampliamente en el uso de la tecnología de Internet para soluciones altamente escalables.
- Es flexible porque proporcionan una jerarquía a los paquetes de las librerías de Java. Por lo tanto, se puede acceder a la información detallada del SNMP usando API de bajo-nivel, o Java Beans de alto nivel para simplificar la programación y adicionar funcionalidades.
- Permite la gestión de red basada en Web, dentro del cual incluye modelos como: SAS *(SNMP Applet Server)* y HTTP que nos permitirá gestionar la

red en Internet o aún mejor, gestionar dispositivos delante de los firewall. El SAS APIs puede también ser extendida para añadir soporte SSL (*Secure Socket Layer*), para la gestión de seguridad.

- Cumple los estándares, así como las especificaciones RFC de Internet como: SNMPv1, SNMPv2c SNMPv3, y la Coexistencia entre SNMPv1, SNMPv2c, SNMPv3 [RFC 2576] y el Filtrado de Notificaciones y Expedidor Proxy [RFC 2573].

- Ofrece un amplio rango de opciones de despliegue que se adecua a las necesidades del mercado, para el desarrollo de aplicaciones de gestión.

- Proporciona un rico soporte para maniobrar y manipular MIBs. El paquete de soporte MIB de AdventNet SNMP API está diseñado para permitir programas Java y tomar ventaja completa de la información contenida en los archivos MIB.

- Las aplicaciones de gestión SNMP desarrolladas con AdventNet permiten la comunicación con los agentes para la recuperación de los datos, las cuales son operaciones síncronas/asíncronas o por sondeo a intervalos regulares. Las respuestas recuperadas utilizan el mecanismo callback (llamadas de retorno).

- La aplicación puede ser utilizada para sondear regularmente al agente, entradas de vigilancia cruzadas, y tomar las acciones apropiada basados en los resultados.

- Permite mostrar los datos en forma de UI (*User Interface*) o no UI. En el caso de no mostrar los resultados en forma de UI, los usuarios tienen la posibilidad de construir sus propios componentes UI.

3.3.2 ARQUITECTURA AdventNet SNMP

Está compuesta de API's de alto y bajo nivel para el desarrollo de dichas aplicaciones (Figura 3.3).

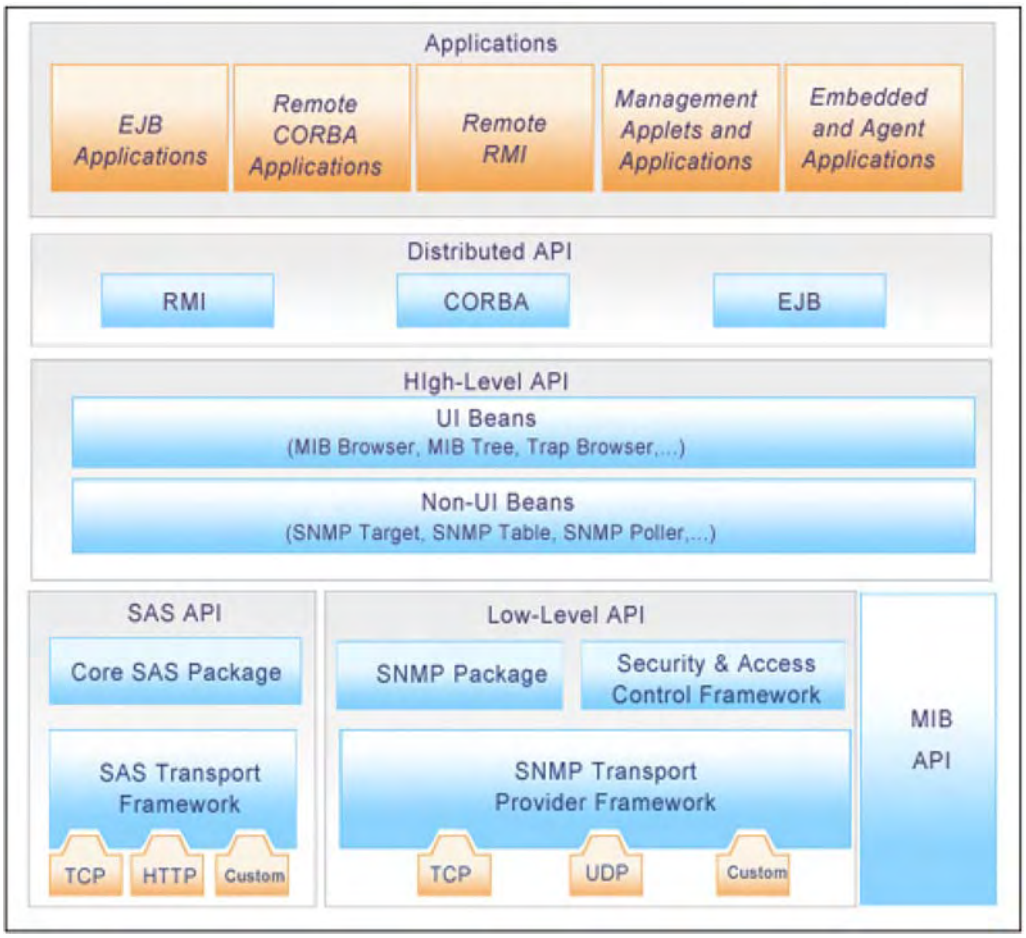


Figura 3.3 API's de AdventNet SNMP [AdventNet04].

A) API de Alto Nivel: Consiste de beans UI y no UI, que permiten construir aplicaciones y applets que incorporen las funciones SNMP proporcionado por el API de bajo nivel.

B) API de bajo nivel: Implementa las funciones de núcleo del protocolo. Incluye clases que facilitan la comunicación con entidades pares SNMP y ofrecer seguridad de mensajes y privacidad a las aplicaciones, también incluye clases que pueden ser utilizados en los applets de gestión ejecutándose desde un browser, soportando comunicación multilingual con los dispositivos.

El API de bajo nivel proporciona la referencia para la implementación de USM (*User Security Model*) y VACM (*View Access Control Model*), así como de los protocolos independientes, los cuales pueden conectarse con el protocolo de transporte.

CAPITULO 4 Creación de la Aplicación de Gestión, Modelamiento y Gestión de Performance

En este capítulo se describirá el desarrollo de la aplicación de gestión de red basado en el agente SNMP. Primero se describirá el diseño del gestor SNMP, el cual ha sido desarrollado tomando como base la arquitectura de Gestión AdventNetSNMP API, cuya organización ha sido indicado en la figura 3.3, para luego entrar al modelamiento de clases y a la medición del performance, a través de expresiones numéricas, los cuales son utilizadas por la aplicación para realizar el proceso de gestión.

4.1 CREACIÓN DE LA APLICACIÓN DE GESTIÓN DE RED

La aplicación a desarrollar es una aplicación de gestión, desarrollado en el lenguaje de programación Java en conjunción con los API's de AdventNet SNMP, cuya estructura está conformada por los siguientes componentes: (Figura 4.1).

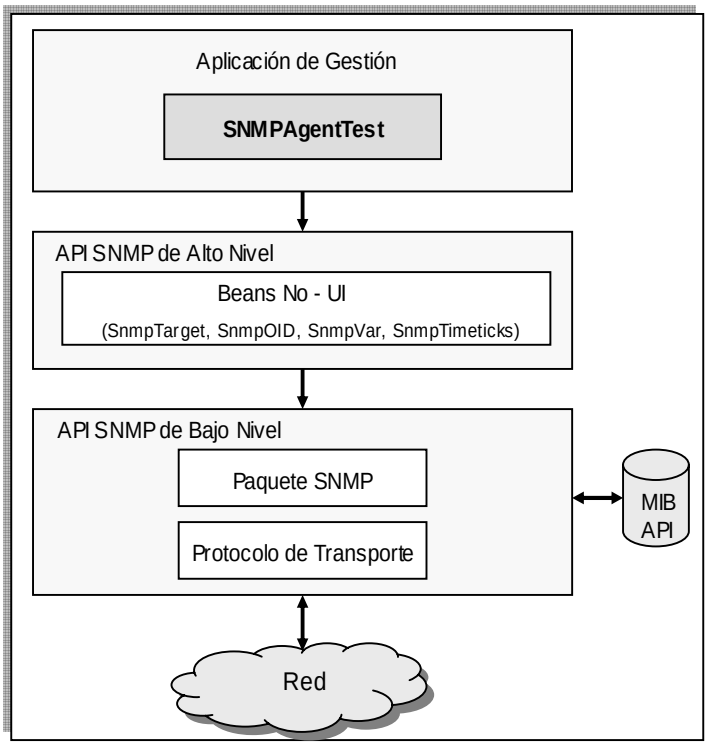


Figura 4.1 Estructura de la Aplicación de Gestión [ELLS].

1. Aplicación de Gestión – representado por el software o programa de gestión denominado "SNMPAgentTest", el cual ha sido desarrollado en Java. Es éste quien se encargará de gestionar y monitorear a las diferentes estaciones dentro de la red.
2. API SNMP de Bajo Nivel – que se encargará de implementar el protocolo SNMPv1, el cual facilitará la comunicación con entidades pares SNMP.
3. API MIB – mediante el cual la aplicación podrá acceder a la información MIB de cada estación gestionada y obtener los valores de las variables para ser evaluadas.

El protocolo de gestión utilizado para el proceso del test a cada una de las estaciones, es el protocolo SNMPv1 el cual está basado en comunidades.

La figura 4.2 muestra el procedimiento o ruta a seguir para el desarrollo de esta aplicación y el código Java desarrollado en el anexo B.

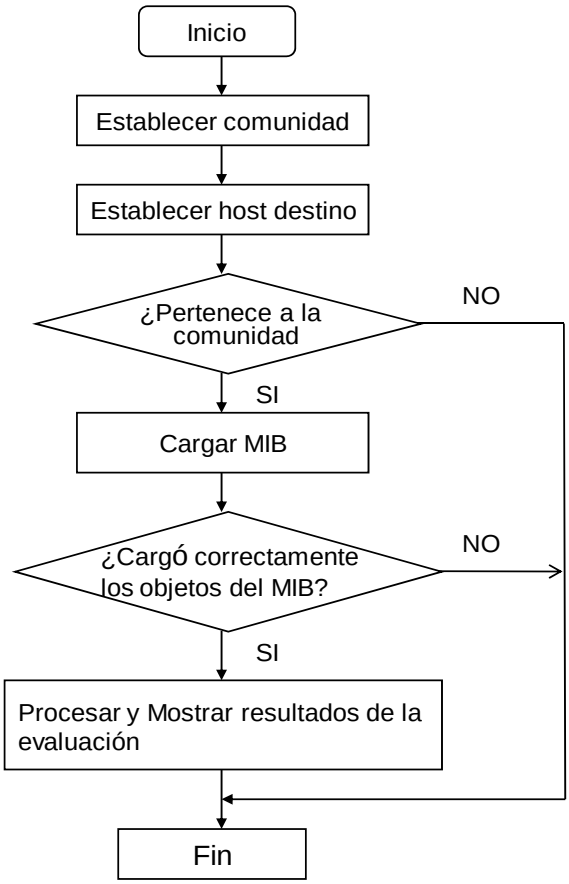


Figura 4.2 Diagrama de flujo de ejecución de la aplicación de gestión [ELLS].

4.2 MODELAMIENTO DE CLASES PARA EL DESARROLLO DE LA APLICACIÓN

Para el desarrollo de la aplicación se utiliza el lenguaje unificado de modelamiento (UML).

A. Diagrama de Clases

Dentro de los API's empleados para el desarrollo de esta aplicación la podemos mencionar (Figura 4.3):

- valueTest(): Método principal de la aplicación de gestión.
- resetSysName(): Método empleado para la limpieza del registro de XP.
- SnmpTarget: Clase que nos permite configurar a cada uno de los dispositivos dentro de la red, a través de los siguientes métodos:

- setTargetHost(): Método que establece el host o recurso a gestionar.
- setWriteCommunity(): Método que establece la comunidad a la que pertenecen cada una de las estaciones a gestionar, y que aceptarán los mensajes, en nuestro caso, la comunidad ha sido denominada “unmsm”.
- loadMibs(): Método que cargará el MIB-1213, desde una ruta específica.
- setObjectID(): Método que establece a cada uno de los objetos dentro del MIB, identificados por un OID.
- snmpGet(): Método que recupera los valores de los objetos OID.

Todos estos métodos y la clase SnmpTarget están contenidos dentro del paquete de AdvenNetSNMP API 4.

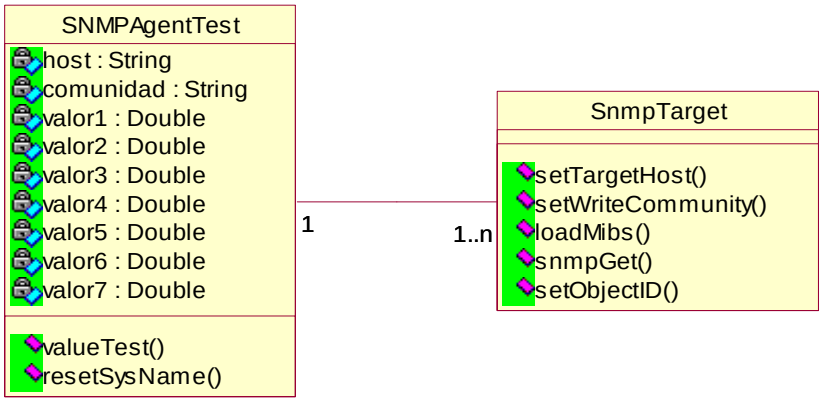


Figura 4.3 Diagrama de clases de la aplicación de gestión SNMP [ELLS].

B. Diagrama de Secuencias

La figura 4.4, describe la secuencia de trabajo de clases que la aplicación de gestión utiliza para crear un objeto para cada dispositivo gestionado, solicitando a la clase SnmpTarget realizar las operaciones de gestión a través de sus diferentes métodos, retornando luego con los resultados y ser mostrado, en un mensaje de texto plano, como DOS de Windows.

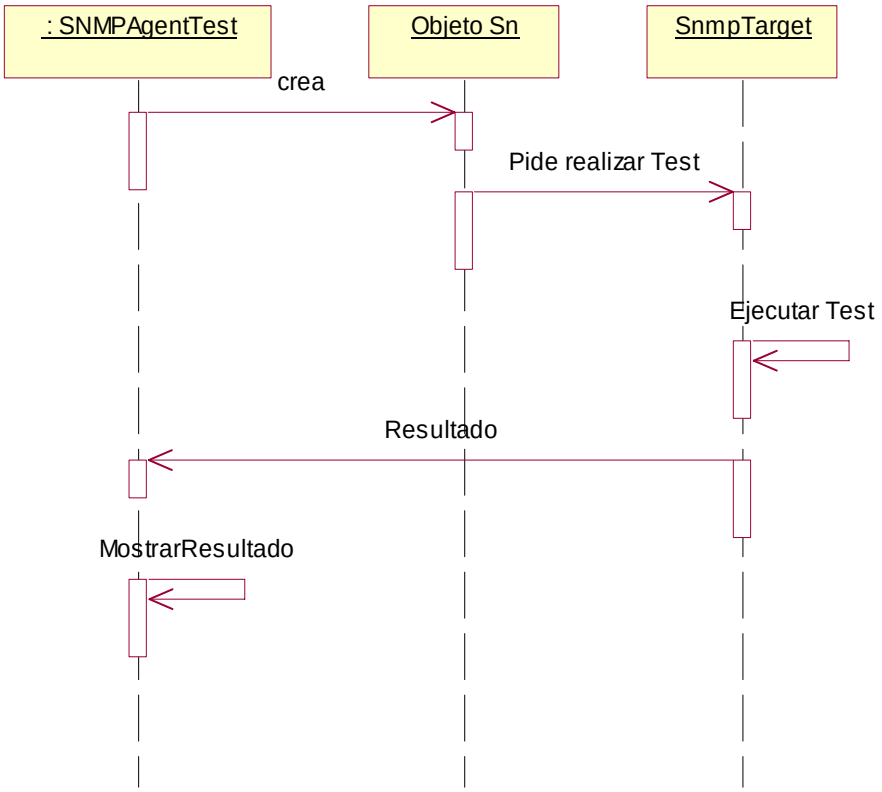


Figura 4.4 Enviando un objeto SNMPAgentTest a una estación a gestionar [ELLS].

C. Diagrama de Actividades

El diagrama de actividades de la figura 4.5, muestra una visión simplificada de lo que ocurre durante una operación ó proceso de gestión, iniciándose con el establecimiento de la comunidad, permitido por el protocolo SNMP v1, al cual se le ha denominado “unmsm”, luego se establece el host a gestionar, así como cargar al MIB de una ruta establecida para la obtención de los valores de los objetos que serán considerados en la evaluación de los parámetros de performance.

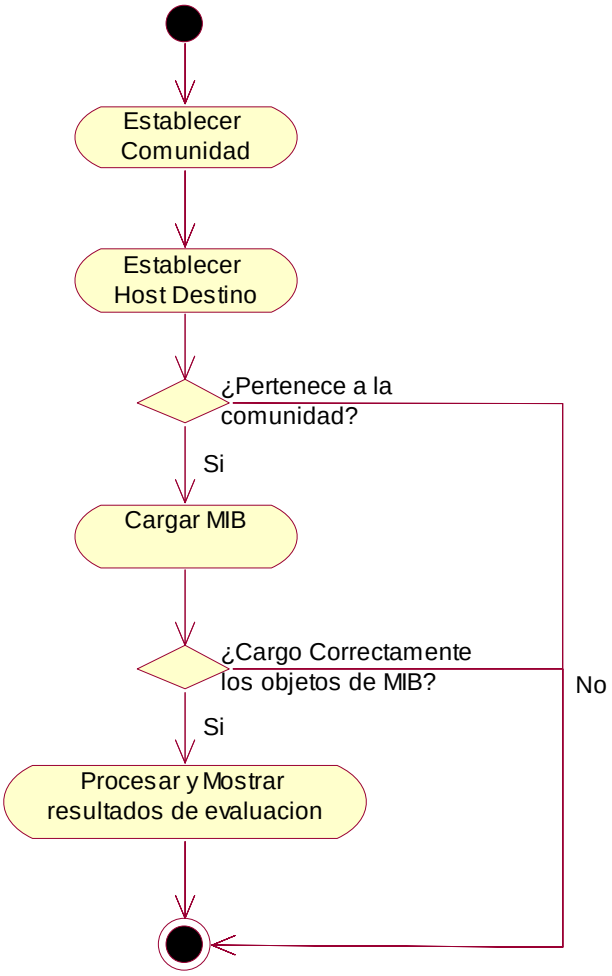


Figura 4.5 Ejecución de la aplicación de gestión [ELLS].

4.3 GESTIÓN DE PERFORMANCE

La gestión de performance implica la optimización del tiempo de respuesta de los servicios de red, vinculando directivas de consistencia de la calidad individual y global de los servicios de red [Cisco03].

4.3.1 MEDICIÓN DEL PERFORMANCE

La función importante en el área de gestión, es la medición del performance, el cual involucra información estadística acerca del tráfico, y los métodos para su reducción, y presentación de los datos [Makki06].

La gestión de performance es un término genérico que incluye la configuración y medición de las distintas áreas de rendimiento como;

Reunir datos de línea de base de la red, medida de la Disponibilidad, medición del Tiempo de Respuesta, medición de la Precisión, medida de la Utilización y la Capacidad de Planificación.

Dentro de la medición de performance, el cual se hizo uso de los objetos del MIB-II para el desarrollo de esta investigación fue: la medición de utilización, la medición de la precisión dentro de determinados enlaces, todo esto expresados a través de expresiones matemáticas, como se mostrará a continuación.

A. UTILIZACIÓN DE LA INTERFAZ

Es la tasa de utilización del enlace. Este valor esta expresado en porcentaje, en la cual el recurso es comparado con su máxima capacidad operacional.
A través de la medida de utilización, los administradores pueden identificar congestión (o potencial congestión) a través de la red. Las variables a utilizarse son mostradas en la siguiente tabla 4.1.

Objetos MIB	**Descripción**
ΔifInOctets	El delta (o diferencia) entre dos ciclos de sondeo recogida del objeto "ifInOctets" snmp, el cual representa el número de octetos entrantes de tráfico.
ΔifOutOctets	El delta entre dos ciclos de sondeo recogida del objeto "ifOutOctets" snmp, el cual representa el número de octetos salientes de tráfico.
ifSpeed	La velocidad de la interfaz reportada del objeto "ifSpeed" snmp, en bits/seg
Número de segundos en Δ (ΔsysUpTime)	El delta de tiempo de duración después de dos ciclos de sondeo del objeto "sysUpTime" snmp, en centésimas de segundos.

Tabla 4.1 Variables MIB para la evaluación de la utilización de la interfaz

Expresión Matemática:

$$\text{Utilización} = \frac{(\Delta\text{ifInOctets} + \Delta\text{ifOutOctets}) \times 8}{(\text{Número de segundos en } \Delta) \times \text{ifSpeed}} \times 100$$

En términos de objetos MIB, será necesario introducir una constante de 8 (considerando un octeto compuesto de 8 bits) y multiplicar por 100 por el porcentaje.

B. PRECISIÓN Y TASA DE ERROR

Precisión es la medida del tráfico de la interfaz que no resulta dentro del error y puede ser expresada en términos de porcentaje, que compara la tasa de un buen resultado a la tasa de paquetes totales durante un periodo de tiempo. Primero será necesario medir la tasa de error. Por ejemplo, si dos de cada 100 paquetes resulta dentro del error, la tasa de error será de 2% y la precisión será de 98%. Dentro de las causas comunes que pueden causar error a la interface podemos mencionar: a las especificaciones del cableado de salida, interfaces eléctrica, y a caídas de hardware y software.

Las variables a utilizar son mostradas en la tabla 4.2.

Objetos MIB	**Descripción**
ΔifInErrors	El delta (o diferencia) entre dos ciclos de sondeo recogida del objeto “ifInErrors” snmp, el cual representa el número de paquetes entrantes con un error.
ΔifInUcastPkts	El delta entre dos ciclos de sondeo recogida del objeto “ifInUcastPkts” snmp, el cual representa el número de paquetes unicast entrantes.
ΔifInNUcastPkts	El delta entre dos ciclos de sondeo recogida del objeto “ifInNUcastPkts” snmp, el cual representa el número de paquetes no-unicast entrantes (multicast y broadcast).

Tabla 4.2 Variables MIB para la evaluación de la precisión y la tasa de error

La fórmula para la tasa de error y la precisión es usualmente expresada en porcentaje.

$$\text{Tasa de Error} = \frac{\Delta\text{ifInErrors}}{(\Delta\text{ifInUcastPkts} + \Delta\text{ifInNUcastPkts})} \text{ X } 100$$

$$\text{Precisión} = 100 - \frac{\Delta\text{ifInErrors}}{(\Delta\text{ifInUcastPkts} + \Delta\text{ifInNUcastPkts})} \text{ X } 100$$

Todas las expresiones descritas son calculadas por la aplicación de gestión.

CAPITULO 5 Evaluación, Implementación y Análisis de la Aplicación de Gestión en el Entorno Centralizado

En este capítulo, se describirá el proceso de evaluación y el análisis cuantitativo de la aplicación de gestión, desde el punto de vista matemático, para luego pasar a su implementación y posteriormente mostrar los resultados obtenidos a través de pruebas realizadas durante el periodo de desarrollo de esta investigación. Dichas pruebas han sido realizadas en lugares de alto tráfico, como es el caso de las cabinas de Internet.

5.1 ANÁLISIS CUANTITATIVO DEL TRÁFICO DE GESTIÓN

Para el desarrollo de este análisis, se tomará en cuenta la transmisión de las cabeceras introducidas por la encapsulación de los protocolos, teniendo como base los niveles de capas del modelo OSI.

La figura 5.1 muestra como un mensaje SNMP es transportado a todas las entidades enviando solicitudes, para luego retornar con su réplica respectiva.

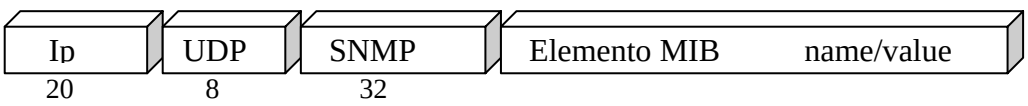


Figura 5.1. Encapsulación de solicitudes y réplicas SNMP [BaldiPicco98].

Con estas consideraciones, podemos establecer que:

1. La cantidad de datos conmutados entre las entidades de capas de red es independiente de la tecnología de las capas más bajas y por lo tanto invariante a través de toda la red.
2. Si un bloque de datos de tamaño X se transmite a nivel de aplicación, entonces la cantidad actual de datos conmutados será: [BaldiPicco98].

$$X' = \alpha(x) + \beta(x)X \qquad (1)$$

Si $\eta(x) = \frac{\alpha(x)}{X} + \beta(x)$ entonces $X' = \eta(x)\, X$ (2)

Donde: $\eta(x) > 1$

$\beta(x)$: Cabecera introducida por la encapsulación del mensaje.

$\alpha(x)$: Información de control durante la fase de organización.

$\eta(x)$: Función de cabecera, desde que da cuenta de la información de control (cabecera de protocolo) añadidas a X.

Considerando un escenario donde la estación de gestión de red (NMS) recupera los datos de gestión desde un conjunto de dispositivos, la red gestionada estará en términos de:

N: Número de dispositivos gestionados.

Q: Número de instrucciones SNMP ejecutadas sobre el MIB.

Iq: Ancho de la instrucción q^{th} , expresado en bytes

Rqn: Ancho de la réplica q^{th} que retorna del nodo n^{th}, expresado en bytes

ηcs: Función de cabecera utilizada al enviar un mensaje de solicitud (request).

$\acute{\eta} cs$: Función de cabecera utilizada en la réplica (reply).

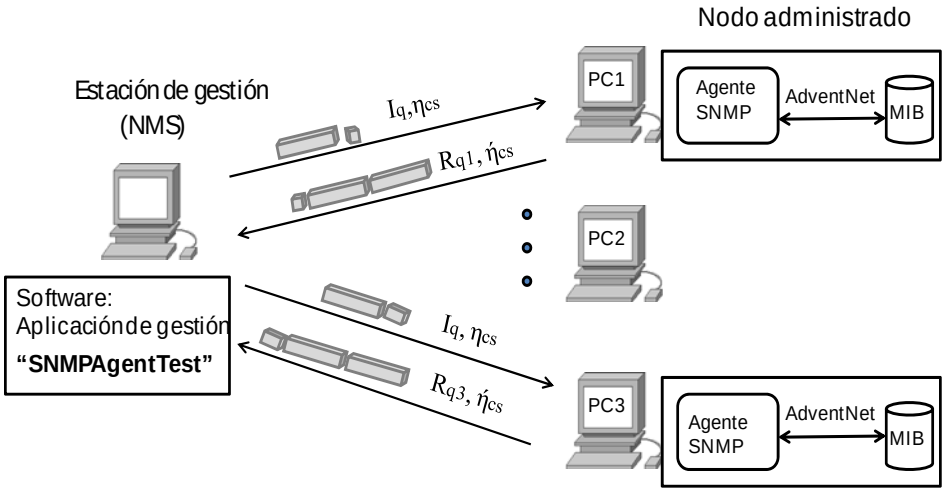


Figura 5.2. Tráfico en el Modelo Centralizado [ELLS].

En la figura 5.2, la aplicación de gestión "SNMPAgentTest" envía una instrucción GET por cada variable MIB solicitada, los cuales son: ifInOctets, ifOutOctets, ifSpeed, sysUpTime, ifInError, ifInUcastPkts, ifInNUcastPkts.

Entonces:

Q = 1……7 (Número de instrucciones SNMP a enviar).

N = 1…...3 (Número de nodos gestionados).

Luego:

El tráfico generado por las estaciones gestionadas viene dado por:

$$Tcs = \eta_{cs}I_1 + \acute{\eta}_{cs}R_{11} + \eta_{cs}I_2 + \acute{\eta}_{cs}R_{21} + \eta_{cs}I_3 + \acute{\eta}_{cs}R_{31} + \eta_{cs}I_4 + \acute{\eta}_{cs}R_{41} + \eta_{cs}I_5 + \acute{\eta}_{cs}R_{51}$$
$$+ \eta_{cs}I_6 + \acute{\eta}_{cs}R_{61} + \eta_{cs}I_7 + \acute{\eta}_{cs}R_{71} +$$
$$\eta_{cs}I_1 + \acute{\eta}_{cs}R_{12} + \eta_{cs}I_2 + \acute{\eta}_{cs}R_{22} + \eta_{cs}I_3 + \acute{\eta}_{cs}R_{32} + \eta_{cs}I_4 + \acute{\eta}_{cs}R_{42} + \eta_{cs}I_5 +$$
$$\acute{\eta}_{cs}R_{52} + \eta_{cs}I_6 + \acute{\eta}_{cs}R_{62} + \eta_{cs}I_7 + \acute{\eta}_{cs}R_{72} +$$
$$\eta_{cs}I_1 + \acute{\eta}_{cs}R_{13} + \eta_{cs}I_2 + \acute{\eta}_{cs}R_{23} + \eta_{cs}I_3 + \acute{\eta}_{cs}R_{33} + \eta_{cs}I_4 + \acute{\eta}_{cs}R_{43} + \eta_{cs}I_5 +$$
$$\acute{\eta}_{cs}R_{53} + \eta_{cs}I_6 + \acute{\eta}_{cs}R_{63} + \eta_{cs}I_7 + \acute{\eta}_{cs}R_{73.}$$

$$T_{cs} = \sum_{n=1}^{3} \sum_{q=1}^{7} (\eta_{cs}I_q + \acute{\eta}_{cs}R_{qn}) \text{ bytes} \qquad (3)$$

Donde:

$\eta_{cs}I_1$: Cantidad de datos enviados en la solicitud 1 a cada uno de los nodos.

$\acute{\eta}_{cs}R_{11}$: Cantidad de datos recibidos de la solicitud 1 del nodo1.

$\eta_{cs}I_2$: Cantidad de datos enviados en la solicitud 2 a cada uno de los nodos.

$\acute{\eta}_{cs}R_{21}$: Cantidad de datos recibidos de la solicitud 2 del nodo1.

Representación general del tráfico generado en un entorno centralizado (representado por el modelo Cliente-Servidor), expresado en bytes.

$$T_{cs} = \sum_{n=1}^{N} \sum_{q=1}^{Q} (\eta_{cs}I_q + \acute{\eta}_{cs}R_{qn}) \qquad (4)$$

5.2 ANÁLISIS CUANTITATIVO DEL TIEMPO DE GESTIÓN

El tiempo de gestión, esta dado en función del costo de la comunicación asociada a los enlaces, asignando un coeficiente de ponderación $0 \leq \theta \leq 1$.

El valor del coeficiente de costo será determinado por el gestor de acuerdo a la noción del costo asociado al enlace, y puede ser una combinación de varios factores. Por ejemplo, un alto costo puede ser debido a alta latencia, retardo, ancho de banda sobre los enlaces o al hecho de cómo un enlace está conectado al NMS [BaldiPicco98] (Figura 5.3).

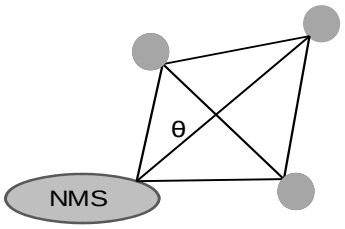


Figura 5.3 Costo de gestión de un enlace [BaldiPicco98].

El mismo coeficiente θ es asociado a cada enlace, y el costo total de la ejecución de una tarea de gestión viene dada por:

$$K = \theta T \qquad (5)$$

Donde: T puede ser la expresión de tráfico.

Antes de hacer el análisis, es necesario escribir los conceptos de retardo (λ) y rendimiento (τ), los cuales son considerados como parámetros para obtener una buena calidad de servicio (QoS), y que han sido utilizados para esta evaluación.

- RENDIMIENTO (Throughput): Especifica cuantos datos son transferidos a través de la red durante un periodo de tiempo. El throughput es medido después de la transmisión de datos y varía con el tiempo debido al tráfico y la congestión.

  Se expresa en bits, kilobits, o megabits por segundos.
- RETARDO (Delay): Se refiere al tiempo que dura en transmitirse un bit desde su origen hasta su destino. Es un parámetro que se emplea para medir el máximo retardo en una red de extremo a extremo. Dentro de los tipos de retardo se pueden mencionar a; *Retardo por procesamiento* en el nodo (tiempo requerido en analizar el encabezado y determinar la salida del enlace), *retardo de colas* (tiempo de espera en el buffer para ser transmitido), *retardo de transmisión* (tiempo requerido para colocar todos los bits en el enlace) y *retardo de propagación* (tiempo transcurrido hasta llegar al final del trayecto físico). Como se puede observar el retardo siempre estará presente, es decir, no puede ser eliminado y puede ser expresado desde unos pocos milisegundos hasta varios cientos de milisegundos en redes IP.

  Se considerará un retardo promedio para el desarrollo de nuestra investigación en todas las expresiones subsiguientes, representado por λ.

5.2.1 MODELO DE COSTO DE GESTIÓN

Del modelo de costo de gestión de [BaldiPicco98], el costo de tiempo que le llevará a la NMS realizar el proceso de gestión, estará dado por la ecuación:

$$Ktcs = \theta T \qquad (6)$$

Donde:

θ: Valor de coeficiente de costo ($\lambda/T + 1/\tau$) de un determinado enlace.

T: Tráfico generado en el enlace.

λ: Retardo

τ: Throughput

Luego, de [Straber]:

$$Ktcs = \lambda + T/\tau \qquad (7)$$

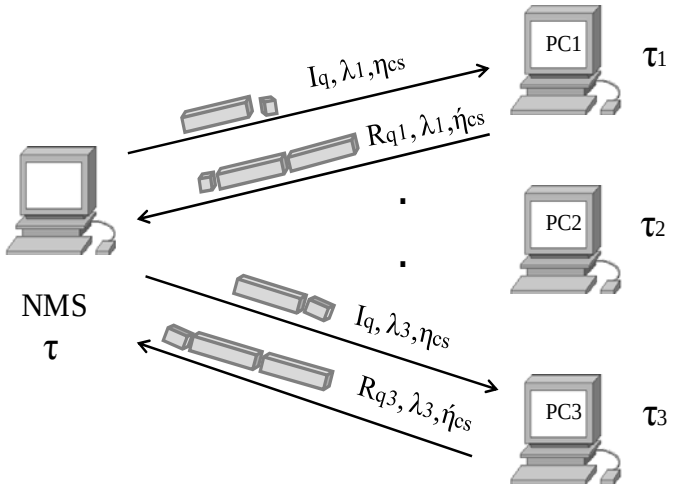


Figura 5.4. Tiempos de gestión del Modelo Centralizado [ELLS].

El tiempo de gestión de la figura 5.4 viene dado por:

A. Entre la NMS y el nodo 1

$$Kt_{cs1} = \lambda_1 + \frac{\eta_{cs}I_1}{\tau_1} + \lambda_1 + \frac{\acute{\eta}_{cs}R_{11}}{\tau} + \lambda_1 + \frac{\eta_{cs}I_2}{\tau_1} + \lambda_1 + \frac{\acute{\eta}_{cs}R_{21}}{\tau} +$$
$$\lambda_1 + \frac{\eta_{cs}I_3}{\tau_1} + \lambda_1 + \frac{\acute{\eta}_{cs}R_{31}}{\tau} + \lambda_1 + \frac{\eta_{cs}I_4}{\tau_1} + \lambda_1 + \frac{\acute{\eta}_{cs}R_{41}}{\tau} +$$
$$\lambda_1 + \frac{\eta_{cs}I_5}{\tau_1} + \lambda_1 + \frac{\acute{\eta}_{cs}R_{51}}{\tau} + \lambda_1 + \frac{\eta_{cs}I_6}{\tau_1} + \lambda_1 + \frac{\acute{\eta}_{cs}R_{61}}{\tau} +$$
$$\lambda_1 + \frac{\eta_{cs}I_7}{\tau_1} + \lambda_1 + \frac{\acute{\eta}_{cs}R_{71}}{\tau} \qquad (8)$$

B. Entre la NMS y el nodo 2

$$Kt_{cs2} = \lambda_2 + \frac{\eta_{cs}I_1}{\tau_2} + \lambda_2 + \frac{\acute{\eta}_{cs}R_{12}}{\tau} + \lambda_2 + \frac{\eta_{cs}I_2}{\tau_2} + \lambda_2 + \frac{\acute{\eta}_{cs}R_{22}}{\tau} +$$
$$\lambda_2 + \frac{\eta_{cs}I_3}{\tau_2} + \lambda_2 + \frac{\acute{\eta}_{cs}R_{32}}{\tau} + \lambda_2 + \frac{\eta_{cs}I_4}{\tau_2} + \lambda_2 + \frac{\acute{\eta}_{cs}R_{42}}{\tau} +$$
$$\lambda_2 + \frac{\eta_{cs}I_5}{\tau_2} + \lambda_2 + \frac{\acute{\eta}_{cs}R_{52}}{\tau} + \lambda_2 + \frac{\eta_{cs}I_6}{\tau_2} + \lambda_2 + \frac{\acute{\eta}_{cs}R_{62}}{\tau} +$$
$$\lambda_2 + \frac{\eta_{cs}I_7}{\tau_2} + \lambda_2 + \frac{\acute{\eta}_{cs}R_{72}}{\tau} \qquad (9)$$

C. Entre la NMS y el nodo 3

$$Kt_{cs3} = \lambda_3 + \frac{\eta_{cs}I_1}{\tau_3} + \lambda_3 + \frac{\acute{\eta}_{cs}R_{13}}{\tau} + \lambda_3 + \frac{\eta_{cs}I_2}{\tau_3} + \lambda_3 + \frac{\acute{\eta}_{cs}R_{23}}{\tau} +$$
$$\lambda_3 + \frac{\eta_{cs}I_3}{\tau_3} + \lambda_3 + \frac{\acute{\eta}_{cs}R_{33}}{\tau} + \lambda_3 + \frac{\eta_{cs}I_4}{\tau_3} + \lambda_3 + \frac{\acute{\eta}_{cs}R_{43}}{\tau} +$$
$$\lambda_3 + \frac{\eta_{cs}I_5}{\tau_3} + \lambda_3 + \frac{\acute{\eta}_{cs}R_{53}}{\tau} + \lambda_3 + \frac{\eta_{cs}I_6}{\tau_3} + \lambda_3 + \frac{\acute{\eta}_{cs}R_{63}}{\tau} +$$
$$\lambda_3 + \frac{\eta_{cs}I_7}{\tau_3} + \lambda_3 + \frac{\acute{\eta}_{cs}R_{73}}{\tau} \qquad (10)$$

La aplicación de gestión “SNMPAgentTest” crea “n” objetos Java para la realización de las operaciones de gestión de manera paralela y simultánea para cada una de las estaciones a gestionar y de acuerdo a su arquitectura Cliente/Servidor el tiempo va ser equivalentemente constante para “n” nodos gestionados.

De las ecuaciones obtenidas $\tau \geq \tau_n$, debido a que el ancho de datos de la réplica en la NMS es mayor que el ancho de las solicitudes en los nodos gestionados.
Para el caso de gestión de una LAN simple, consideramos solo un enlace entre la NMS y un nodo gestionado cualesquiera y valores promedios ($\lambda_n = \lambda_{cs}$ y $\tau = \tau_n = \tau_{cs}$) para nuestro análisis:
Luego:

$$Kt_{cs} = 14\lambda_{cs} + \sum_{q=1}^{7} \frac{\eta_{cs}I_q}{\tau_{cs}} + \sum_{q=1}^{7} \frac{\acute{\eta}_{cs}R_q}{\tau_{cs}} \qquad (11)$$

Representación general:

$$Kt_{cs} = 14\lambda_{cs} + \sum_{q=1}^{Q} \frac{\eta_{cs}I_q + \acute{\eta}_{cs}R_q}{\tau_{cs}} \qquad (12)$$

Donde:

λ_{cs}: Retardo de enlace (entre la NMS y el nodo gestionado).

τ_{cs}: Throughput entre la NMS y el nodo gestionado (solicitud y réplica).

Q: Número de instrucciones SNMP.

η_{csIq} : Cantidad de datos de la instrucción q^{th} , incluyendo las cabeceras.

$\acute{\eta}_{csRq}$: Cantidad de datos de la réplica de la solicitud q^{th} , incluyendo cabeceras.

Estableciendo las unidades de la expresión (12), donde cada uno de los términos están expresado en bytes (1byte = 8 bits) tendremos:

$$Kt_{cs} = 112\lambda_{cs\ seg} + 8\sum_{q=1}^{Q} \frac{\eta_{cs}I_q + \acute{\eta}_{cs}R_q}{\tau_{cs}}\ _{seg} \qquad (13)$$

5.3 IMPLEMENTACIÓN

Para el desarrollo de esta investigación las herramientas de trabajo utilizados, tanto software como hardware son de uso comercial y es muy utilizado por cualquier usuario común, y disponible en el mercado comercial.

5.3.1 PROCEDIMIENTO

Dentro de las especificaciones técnicas para la realización del test se consideraron:

Computadores: Pentium IV, Pentium CoreDuo.

Softwares: S.O Windows XP, Java SE 5.0, y AdventNetSNMP 4.

Red: Ethernet, switch D-Link, MODEM/Router.

El procedimiento a seguir fue:

1. Instalar el agente SNMP tanto en la estación de gestión, como en los nodos o estaciones a gestionar.
2. Habilitar el servicio SNMP, estableciendo la comunidad READ/WRITE a “unmsm” en todas las PC’s.
3. Ejecutar la aplicación de gestión desde la estación de gestión, hacia todas las PC’s mediante sondeos.

Todo el procedimiento de instalación realizado se describe en el anexo A.

5.3.2 EJECUCIÓN

Se ejecuta desde el comando de línea, dentro del símbolo de sistema proporcionado por Windows.

C:\> java SNMPAgentTest

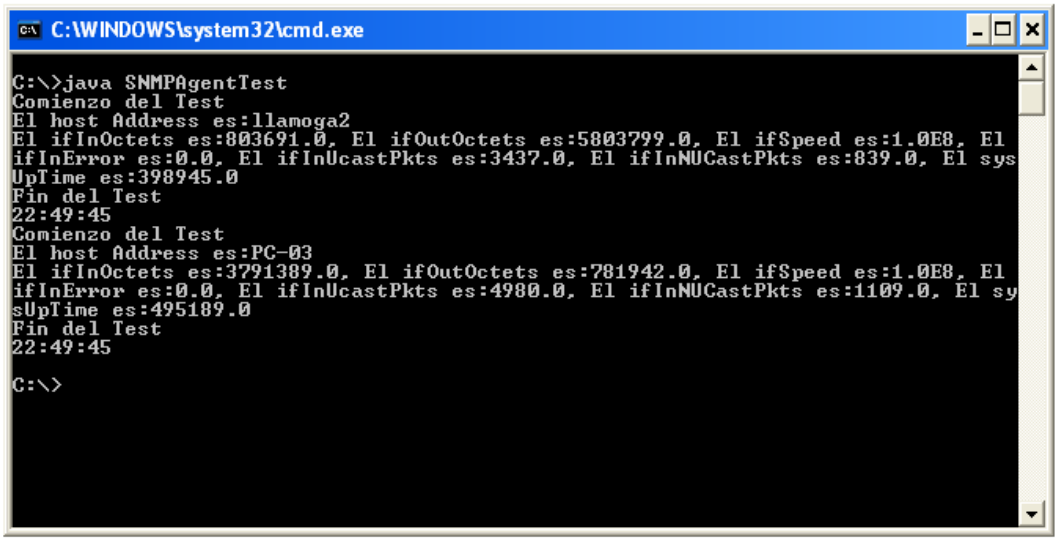


Figura 5.5 .Obtención de datos en el modelo centralizado [ELLS].

Como se puede observar, en el figura 5.5, son los resultados del test realizado por la aplicación de gestión "SNMPAgentTest", tanto a la estación de gestión (llamoga2) así como a una de las estaciones gestionadas (pc-03).

5.4 RESULTADOS

- El proceso desarrollado dentro del entorno centralizado, se ha considerado tanto al gestor (NMS), y a las estaciones gestionadas (PC's).

A. UTILIZACIÓN DE LA INTERFAZ:

Este parámetro de performance nos permite determinar cómo está trabajando la interfaz, y en caso de ocurrir una sobre utilización nos permitirá que hay más tráfico en la cola para pasar sobre la interfaz que la interfaz puede manejar (Cisco03), y por consiguiente a una caída en la red. Los valores de medición de los objetos MIB se muestran en las tablas 5.1, 5.2 y 5.3.

OBJETOS MIB	t 1		t2	
	ESTACION DE GESTIÓN (NMS)	ESTACIÓN GESTIONADA	ESTACION DE GESTIÓN (NMS)	ESTACIÓN GESTIONADA
ifInOctets	803691	3791389	965097	5050354
ifOutOctets	5803799	781942	7019818	995977
ifSpeed	1.00E+08	1.00E+08	1.00E+08	1.00E+08
sysUpTime	3989.45	4951.89	4615.04	5577.46
ifInErrors	0	0	0	0
ifInUcastPkts	3437	4980	4102	6472
ifInNUcastPkts	839	1109	932	1190

Tabla 5.1 Valores de objetos MIB de la NMS y una estación gestionada en el tiempo t1 y t2 en el entorno centralizado

OBJETOS MIB	t 1			t2		
	ESTACION DE GESTIÓN (NMS)	ESTACIONES GESTIONADAS		ESTACION DE GESTIÓN (NMS)	ESTACIONES GESTIONADAS	
ifInOctets	1122983	6322812	11970784	1410223	7598607	13987611
ifOutOctets	8231805	1231197	1719318	10633745	1459655	1865533
ifSpeed	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08
sysUpTime	5380.82	6343.1	10092.04	6128.23	7090.51	10839.46
ifInErrors	0	0	0	0	0	0
ifInUcastPkts	4733	8059	13079	5910	9649	14202
ifInNUcastPkts	1034	1282	2308	1134	1369	2402

Tabla 5.2 Valores de objetos MIB de la NMS y dos estaciones gestionadas en el tiempo t1 y t2 en el entorno centralizado.

OBJETOS MIB	t 1				t2			
	ESTACION DE GESTIÓN (NMS)	ESTACIONES GESTIONADAS			ESTACION DE GESTIÓN (NMS)	ESTACIIONES GESTIONADAS		
ifInOctets	1717253	8952332	15224588	27836332	2147787	10272055	16453268	29410088

ifOutOctets	13043348	1766829	2020442	41310566	16637871	2052904	2174749	41972985
ifSpeed	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08
sysUpTime	7237.62	8199.87	11948.85	14119.53	8193.42	9155.68	12904.67	15075.14
ifInErrors	0	0	0	0	0	0	0	0
ifInUcastPkts	7166	1177.5	15399	98646	9060	13799	16593	103128
ifInNUcastPkts	1356	1573	2618	3551	1487	1649	2742	3681

Tabla 5.3 Valores de objetos MIB de la NMS y tres estaciones gestionadas en el tiempo t1 y t2 en el entorno centralizado

La utilización de la interfaz después de cada operación de realizada entre:

1. El gestor y una estación gestionada.
2. El gestor y dos estaciones gestionadas.
3. El gestor y tres estaciones gestionadas.

Se muestra en la siguiente tabla 5.4

UTILIZACION DE LA INTERFAZ EN EL ENTORNO CENTRALIZADO		
N° DE MAQUINAS (PC´s)	ESTACION DE GESTION (%)	ESTACIONES GESTIONADAS (%)
2 (NMS - 1PC's)	1.76143E-02	1.88372E-02
3 (NMS - 2PC's)	2.87839E-02	1.61009E-02
	2.87839E-02	2.31520E-02
4 (NMS - 3PC's)	3.36895E-02	1.34403E-02
	3.36895E-02	1.15752E-02
	3.36895E-02	1.87203E-02

Tabla 5.4 Tabla de la utilización de la interfaz en el entorno centralizado

Estos valores fueron próximamente muestreados (Figura 5.6).

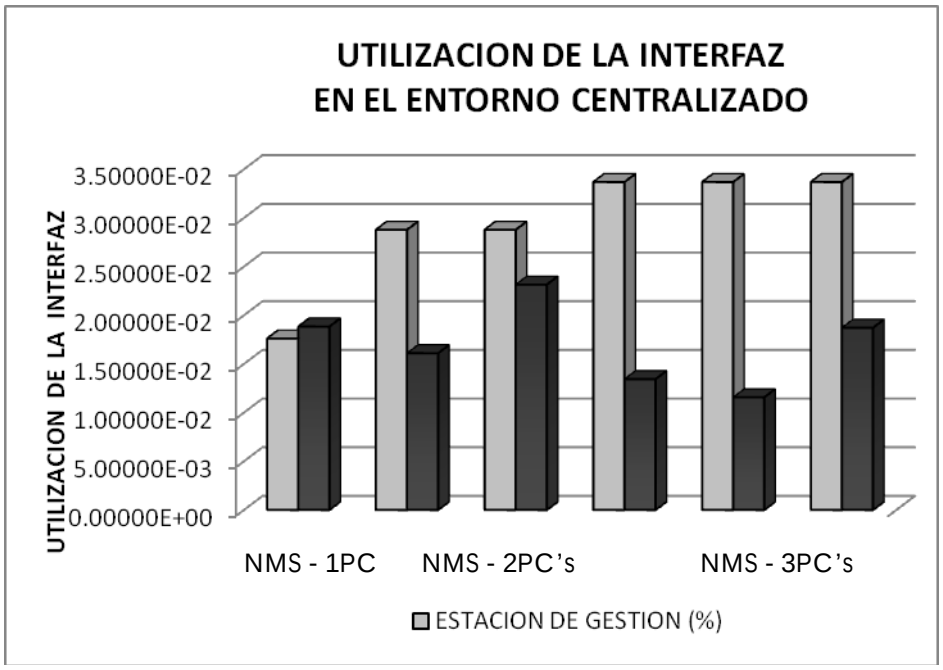


Figura 5.6 Gráfica de la utilización de la interfaz en el entorno centralizado [ELLS].

B. TIEMPOS DE GESTIÓN

Es el tiempo empleado por la estación de gestión (NMS) en realizar el proceso de gestión.

TIEMPOS EN EL ENTORNO CENTRALIZADO	
N° MAQUINAS (PC´s)	TIEMPO (seg)
2 (NMS - 1PC's)	5
3 (NMS - 2PC's)	4
4 (NMS - 3PC's)	5

Tabla 5.5 Tabla de los tiempos de respuesta en el entorno centralizado

La tabla 5.5 muestra los tiempos expresados en segundos después de cada operación de gestión realizada entre:

1. El gestor y una estación gestionada.
2. El gestor y dos estaciones gestionadas.
3. El gestor y tres estaciones gestionadas.

La gráfica 5.7 muestra los valores obtenidos.

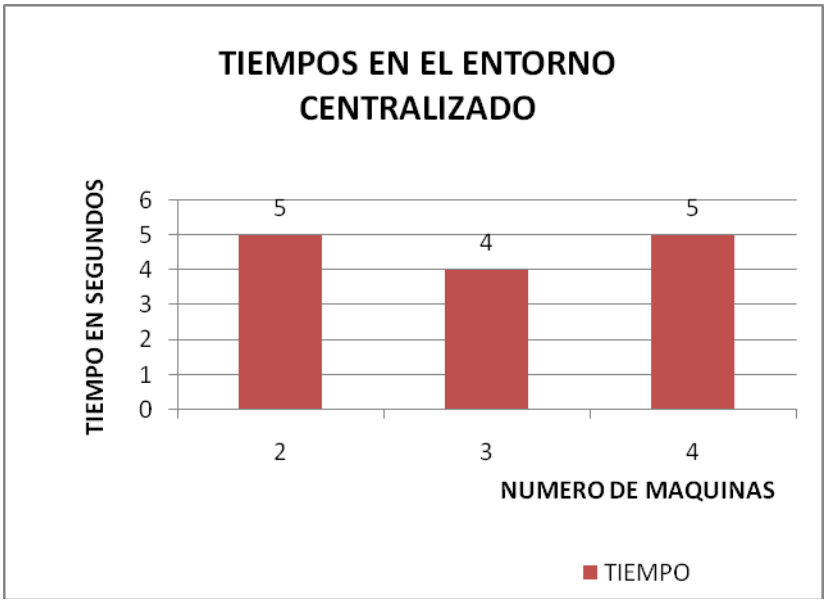


Figura 5.7 Gráfica de tiempos de respuesta en el entorno centralizado [ELLS].

- Del gráfico de los tiempos, son valores obtenidos en diferentes escenarios con cada una de las entidades a gestionar.
- Por otro lado para la evaluación de la tasa de error, no se pudieron graficar, esto es debido, a que la evaluación da como resultado cero, no obteniéndose resultado alguno.
- El porcentaje de utilización de la interfaz en la NMS y en las entidades gestionadas disminuye de acuerdo a la cantidad de estaciones añadidas.
- El tiempo permanece casi constante, tanto en la evaluación con una estación, dos estaciones, tres estaciones, y así sucesivamente.

CAPITULO 6 Agentes Móviles

Los sistemas de agentes móviles han recibido una atención importante de muchos investigadores durante los últimos años como un nuevo paradigma dentro de la programación orientada a objetos, así como de su aplicación de esta tecnología en el mercado de las telecomunicaciones.

En este capítulo introducimos los conceptos de agente, y de agente móvil, describiendo sus ventajas o beneficios, sobre otros sistemas que están siendo utilizados.

Al finalizar este capítulo tendremos un conocimiento completo de ambos conceptos y el porqué de la utilización dentro de esta investigación. Antes de continuar con la tecnología de agentes móviles, permítame describir que es un agente.

6.1 AGENTE

Dada la multiplicidad de roles que un agente puede desempeñar, existen muchas definiciones de ‘agente’, reflejada desde el punto de vista de cada investigador.

He aquí algunas opiniones vertidas por algunos de ellos:

[Maes]: “Un agente es un sistema que trata de cumplir un conjunto de metas dentro de un entorno dinámico y complejo.”

[Shoham]: “Un agente es una entidad cuyo estado es vista como la consistencia de componentes mentales como: creencias, aptitudes, preferencias y compromisos”.

[Hayes-Roth]: “Los agentes inteligentes continuamente ejecutan tres funciones: percepción de las condiciones dinámicas dentro del entorno; acciones que afectan las condiciones del entorno; y razonamiento para interpretar las percepciones, salvar problemas, deducir inferencias y determinar acciones.”

[Wooldridge]: “Es un hardware o más comúnmente un sistema de software de computador que posee las siguientes propiedades: autonomía, habilidad, reactividad y pro-actividad”.

Autonomía - los agentes operan sin la intervención directa de personas u otros, y tienen algún tipo de control sobre sus acciones y estado interno.

Habilidad social - los agentes interactúan con otros agentes (posiblemente humanos) vía algún tipo de lenguaje de comunicación de agentes.

Reactividad - los agentes perciben el entorno o ambiente y responden rápidamente a los cambios que ocurren en dicho entorno.

Pro-actividad - los agentes no actúan simplemente en respuesta a su entorno, sino que son capaces de exhibir ‘comportamientos dirigidos hacia el objetivo’, tomando la iniciativa.

OMG (Object Management Group): “Un agente es un programa de ordenador que actúa autónomamente en nombre de una persona u organización”.

Pero ¿qué es un agente? La respuesta a esta pregunta es compleja, ya que los investigadores utilizan distintas acepciones de dicho término y no existe una definición académica ampliamente aceptada, al igual que tampoco existe una definición exacta para el término inteligencia artificial. Según el Diccionario de la Lengua Española: "Agente: (del lat. agens, -entis, p. a. de agere, hacer). “Que obra o tiene virtud de obrar. …Persona o cosa que produce un efecto…Persona que obra con poder de otro…".

Por supuesto, hay una multitud de opiniones vertidas, pero podemos sacar una conclusión de las características que cumple un agente.

- Movilidad: Capacidad de transportarse de una máquina a otra.
- Reacción: Actuación sobre el entorno mediante un comportamiento estímulo/respuesta.
- Pro-acción: Toma la iniciativa para alcanzar sus objetivos.
- Sociabilidad o Cooperación: Capacidad de comunicarse con otros agentes, programas o personas.
- Aprendizaje o adaptación: Comportamiento basado en la experiencia previa.
- Continuidad temporal: Ejecución contínua en el tiempo.
- Carácter: Inclusión de estados de creencia, deseo e intención (modelo BDI).

6.2 AGENTE MÓVIL

“El interés por los agentes móviles no es motivado por la tecnología, pero si por los beneficios que estos ofrecen” [LangeOshima98].

6.2.1 BENEFICIOS

- Los agentes móviles reducen el tráfico en la red, debido al empaquetamiento de una conversación y despacharlo a un host destino, donde las interacciones pueden realizarse localmente.

También reducen el flujo de registros de datos, en que grandes cantidades de volúmenes de datos almacenados en servidores remotos, deben ser procesados en la localidad de los datos, en vez de transferirlos a través de la red. El fin es simple; mover la computación a los datos en vez de los datos a la computación.

- Se sobreponen a la latencia de red en sistemas de tiempo real críticos, por lo que es importante que las instrucciones de control se ejecuten sin retraso, ya que pueden resolver los problema al ser despachados desde un controlador central e instalarse en los componentes controlados.
- Permiten el encapsulado de protocolos, los cuales permitirán moverse al host remoto para establecer "canales" con los protocolos propietarios.
- Se ejecutan de forma asíncrona y autónoma, en donde las tareas deben ser encajonadas y despachadas hacia la red. Después de ser despachados, los agentes móviles llegan a ser independientes de su proceso de creación y pueden operar de forma asíncrona y autónoma.
- Se adaptan dinámicamente, ya que tiene la habilidad de percibir su entorno y reaccionar de forma autónoma a cambios. Los agentes móviles poseen la capacidad de distribuirse entre los host de una red, manteniendo su configuración al resolver un problema particular.
- Son heterogéneos por naturaleza, debido a que la computación en red desde la perspectiva de hardware y software es fundamentalmente heterogéneo. Los agentes móviles son generalmente independientes del ordenador y de la capa de transporte, y dependientes de su entorno de ejecución.
- Son robustos y tolerantes a fallas, debido a que pueden reaccionar dinámicamente a situaciones y a eventos desfavorables, lo que hace más fácil construir sistemas distribuidos robustos y tolerantes a fallas. Si un servidor está siendo apagado, todos los agentes que se ejecutan en ese servidor serán avisados y se les dará tiempo suficiente para que se envíen a sí mismo y continúen sus operaciones en otro servidor de la red.

El paradigma de movilidad, cuya filosofía principal es llevar el procesamiento a los datos en lugar de los datos al procesamiento [Schoeder00], introduce nuevas funciones como migración, clonación y varios esquemas de interacción. Migrar un

agente consiste en enviar su código y su estado al host remoto, de tal forma que realice una determinada tarea y retornar con el dato requerido, al mismo punto en el que se encontraba antes de migrar. La clonación está referida a la creación de un nuevo agente en el sistema por el agente existente para la distribución y paralelización de tareas a ejecutarse.

La complejidad que presenta la migración del estado y de un código en ejecución se ha abordado de diferentes formas, en concreto se pueden diferenciar dos grandes grupos, los que soportan [Tarr00]:

Movilidad fuerte: se migra el código y estado, abarcando tanto el espacio de datos como el estado de ejecución.

Movilidad débil: se migra el código y el espacio de datos del código, pero no el estado de ejecución.

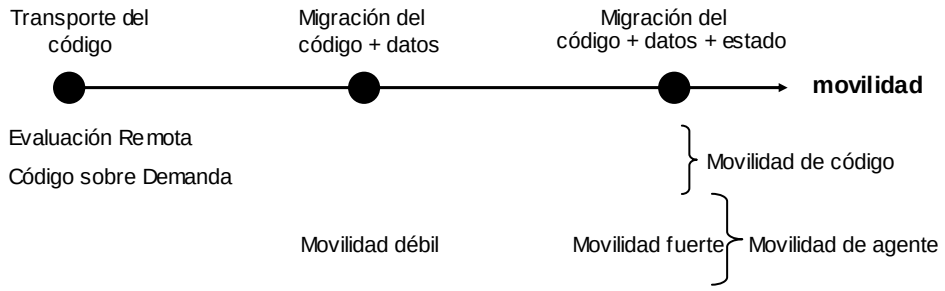


Figura 6.1 Clasificación de los paradigmas de movilidad [ELLS].

La movilidad es una propiedad ortogonal en los agentes, es decir no todos los agentes son móviles. Dentro del estudio de los agentes podemos encontrar que existen dos tipos de agentes:

Agentes estacionario - se ejecutan sólo en el sistema en el cual inician su ejecución, si estos agentes necesitan información contenida en otro sistema o comunicarse con otro agente, utilizan los mecanismos de comunicación remota, tal como las llamadas a procedimientos remotos.

Agentes móviles - no están acotados por el sistema en el cual inician su ejecución, son libres de transportarse a través de la red, creando un ambiente de ejecución, donde transportan su estado y código a otro ambiente de ejecución en la red.

6.3 COMO SE MUEVE UN AGENTE

El proceso para transferir un agente de un sistema a otro se realiza en dos fases.

A. Iniciación de la transferencia.

1. El agente identifica el destino deseado, realiza una petición de viaje al sistema y si es aceptada recibe el permiso para ejecutar la transferencia.
2. El sistema suspende la ejecución del agente e identifica el estado y las partes del agente que serán enviadas.
3. Se realiza la conversión en serie del código y del estado del agente (serialización) y se codifica según el protocolo establecido.
4. El sistema hace la autenticación del agente.
5. Se realiza la transferencia.

B. Recepción del agente.

1. El sistema destinatario acredita al agente.
2. Se realiza la decodificación del agente y la conversión de serie a código y estado del agente (deserialización).
3. El sistema crea la instancia del agente, restaura su estado y continúa la ejecución.

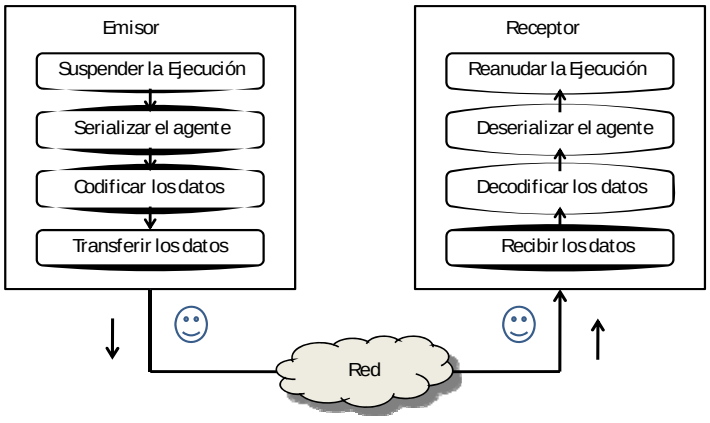


Fig. 6.2 Transferencia de un agente móvil [ELLS].

6.4 SEGURIDAD

El control de la seguridad es uno de los grandes problemas que sufre un agente, ya que un agente es un programa que viaja de un ordenador a otro, al igual que un

virus. Los sistemas deben hacer hincapié en la seguridad de los ordenadores y de la propia agencia [Gómez99]. Dentro de los aspectos típicos de seguridad que deben ser controlados podemos mencionar a: la protección de la máquina contra los agentes, protección contra otros agentes, protección de los agentes contra la máquina y a la protección de la red.

6.4.1 ESTRATEGIAS DE SEGURIDAD

Tanto los sistemas como los propios agentes móviles deben reforzar las tareas de seguridad para evitar, de un modo fiable, los ataques que puedan sufrir. Podemos establecer diferentes políticas de seguridad que nos permitan un buen transporte de información por el agente como: comprobar las credenciales de los participantes en cualquier comunicación, restringir o garantizar las operaciones que puede ejecutar un agente, y gestionar privilegios de acceso a los recursos y establecer límites de consumo. Los requisitos que deben garantizarse en cualquier comunicación son:

Confidencialidad - evitar la escucha del canal.

Integridad - comprobar que los datos no hayan sido modificados durante la transferencia.

Autenticación - tanto el agente – o sistema– emisor como el receptor deben ser identificados para evitar accesos a la información o a recursos reservados.

6.5 LENGUAJES DE COMUNICACIÓN ENTRE AGENTES

Los agentes que forman un sistema pueden colaborar entre ellos para alcanzar sus tareas respectivas. Dicha colaboración se realiza mediante un lenguaje de comunicación comprensible por los agentes del sistema y por otros programas. En contraste con el de un objeto convencional que ejecuta métodos de otro objeto siempre que tenga permiso, un agente puede rechazar una petición de otro agente por lo que deben ser capaces de hablar entre sí, para decidir qué acción realizar o qué datos obtener.

Históricamente, la comunicación entre agentes de un sistema se realizaba mediante lenguajes propios de cada vendedor, impidiendo la comprensión entre los agentes de sistemas heterogéneos, con una semántica informal y con poca autonomía.

Dentro de los lenguajes de comunicación de agentes se pueden mencionar a:

KQML (*Knowledge Query Manipulation Language*): Desarrollado a principios de los años 80, por DARPA el cual pretendía convertirse en una norma para la comunicación entre agentes. Dentro de sus principales características tenemos que es independiente de la sintaxis y la ontología que utiliza. Así mismo, es independiente del mecanismo de transporte (TCP/IP, SMTP, IIOP) e independiente del lenguaje de contenido (KIF, SQL, Prolog, etc).

La Fundación para los Agentes Físicos e Inteligentes (FIPA): Propuso como norma a principios de los años 90 el lenguaje ARCOL (*ARtimis COmmunication Language*). ARCOL conserva al igual que KQML una sintaxis similar al Lisp, pero incluye una semántica formal, lo que proporciona una base rigurosa para la interoperación entre agentes y evita la proliferación de dialectos. Sin embargo, ARCOL carece aún de algunos aspectos deseables para un lenguaje de comunicación de agentes, como permitir una mayor autonomía, una mayor heterogeneidad y el uso de dialectos abiertos.

Luego la organización internacional FIPA desarrolló un lenguaje estandarizado para la comunicación de agentes denominado ACL– FIPA (*Agent Communication Language* FIPA) [FIPA061], cuyo principal objetivo fue darle un sentido semántico a los mensajes que intercambian los agentes. Para lograrlo se basaron en los actos comunicativos, los cuales se basan principalmente en la solicitud de una acción al agente.

FIPA es una organización de estándares de la IEEE Computer Society, que promueve la tecnología basado en agentes y la interoperabilidad de estos estándares con otras tecnologías. Las especificaciones FIPA[1] representan una colección de estándares que promueven la interoperabilidad y los servicios que ellos pueden representar.

6.6 PLATAFORMAS DE AGENTES MOVILES

[1] www.fipa.org/

Todo agente móvil requiere de un entorno especial para su ejecución. Ese entorno es lo que se denomina plataforma. La plataforma además de aportar la capacidad de ejecuciones apropiadas, debe de proporcionar una serie de servicios básicos como:

Movilidad: Que facilite la migración de agentes entre nodos remotos.

Comunicaciones: Que facilite la comunicación de los agentes con el mundo exterior, principalmente, que le permita interactuar con otros agentes en tareas cooperativas.

Nombrado: Que permita nombrar a los agentes de manera que puedan ser identificados de forma univoca, y también nombrar a las plataformas y nodos a los que migran los agentes.

Descubrimiento y localización: Que facilite a los agentes descubrir otros agentes y plataformas con los que puede interactuar, o a las que pueda migrar, para alcanzar los objetivos que tienen encomendados.

Seguridad: Que garantice, por una parte la protección de los agentes ante ataques del host en el que se ejecuta y por otra, la protección del host ante los ataques de los agentes que se ejecuten en él.

Estas son los principales criterios que hay que tener en cuenta al realizar una evaluación de una plataforma [Nguyen02]. La siguiente figura 6.3, muestra como está compuesta una plataforma.

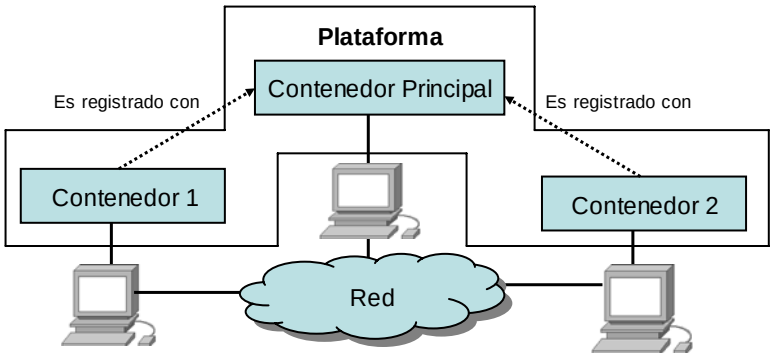


Figura 6.3 Plataforma y contenedores [ELLS].

Existen diversas plataformas de agentes móviles, la mayoría desarrolladas en Java, entre las que cabe destacar:

Agent Development Kit (ADK)

Es una plataforma de agentes basada en Java que incorpora movilidad, seguridad y servicio de directorio[2], a través de sus dos módulos, el ARE (*Agent Runtime Environment*) que es donde habitan los agentes, y el AFC (*Agent Foundation Class*) que proporciona la interfaz entre los agente y el ARE.

Es comercial, pero paralelamente la misma empresa (Tryllian) mantiene un desarrollo libre. No está basada en los estándares FIPA.

Aglets

Es la plataforma de agentes móviles creada por IBM[3], considerada como un reflejo del modelo Apple en Java. Presenta movilidad en sus dos formas (activo y pasivo). En el modelo activo, un aglet es empujado hacia el host remoto (dispatching), y en el modelo pasivo el host remoto jala un aglet del host actual (retracting). Un aglet se extiende de Java el soporte de movilidad débil donde las unidades ejecutantes (código, estado de ejecución) son subprocesos dentro de un interpretador Java, constituyendo así el entorno computacional. Se trata de una plataforma muy conocida. No sigue los estándares FIPA.

Ajanta

Es una plataforma basada en Java que se caracteriza por aportar un tipo de itinerario diferente como por ejemplo bucles. No sigue los estándares FIPA y se encuentra poca activa[4].

Concordia

Es una plataforma basada en Java que se caracteriza por aportar seguridad y control de acceso, permitiendo a los agentes interactuar, modificar estados

[2] http://www.tryllian.org.

[3] http://sourceforge.net/projects/aglets/.

[4] http://www.cs.umn.edu/ajanta.

externos (por ejemplo una base de datos), así como el estado interno del agente[5]. Concordia proporciona un soporte de agentes persistentes, gracias a su rápida recuperación, y así garantizar la transmisión de los agentes a través de la red, pero su principal innovación es que aporta la inclusión del itinerario de los agentes. Es una plataforma comercial y no sigue los estándares FIPA.

D'Agents

Se trata de una plataforma de agentes móviles, antiguamente conocida como AgentTCL, desarrollada sobre el lenguaje TCL, implementa migración débil y fuerte. Utiliza los RPC (Remote Procedure Call) para comunicarse entre plataformas. Incorpora métodos de control de acceso basados en criptografía de llaves públicas. No siguen los estándares FIPA. Es de dominio público, pero actualmente su proceso de desarrollo está inactivo.

FIPA-OS

Es una herramienta de trabajo basado en componentes de desarrollo rápido que cumple las especificaciones FIPA. FIPA-OS[6] soporta la mayoría de las especificaciones experimentales de FIPA y está siendo utilizada continuamente como un proyecto de comunidad open-source, soporta la plataforma Java, y tiene la potencialidad de soportar agente móviles (como una plataforma FIPA), pero la movilidad ha sido implementada solo como un prototipo.

Su política de seguridad está basada en RMI sobre SSL (*Secure Socket Layer*) que soporta seguridad en intra-plataforma y de comunicaciones homogéneas en inter-plataformas.

Grasshopper

Es una plataforma de agentes móviles comercial que se caracteriza por estar basada en los estándares OMG MASIF. Permite la integración del paradigma tradicional cliente-servidor y la tecnología de agente móvil[7]. Su entorno

[5] http://www.merl.com/HSL/Projects/concordia/.

[6] http://www.nortelnetwork.com/fipa-os.

[7] http://www.grasshopper.de/.

distribuido está compuesto de una '*región*' que facilita la gestión de los componentes distribuido, la '*agencia*' que viene a ser una región específica registrada en la plataforma, y de '*place*', en donde reside el agente. Las últimas versiones cumplen las especificaciones FIPA.

JADE

Es un middleware que simplifica el desarrollo de aplicaciones como: la aplicación distribuida, compuestas de entidades autónomas que necesitan comunicarse y colaborar para lograr el funcionamiento del sistema entero, en particular cuando es aplicado al entorno móvil da lugar a una nueva tendencia de evolución que se hace llamar el dispositivo inteligente (smart device), interconexión inteligente (smart inter-conection) [Juarez05]. Más adelante se describirá, con más detalles a este middleware[8].

Semoa

Es un proyecto open- source (Fraunhofer Society - Alemania), es relativamente nuevo que se distribuye bajo licencia LGPL. Básicamente se trata de una especie de kit de herramientas para el desarrollo de agentes móviles. Su característica principal es el enfoque fuerte en el aspecto de la seguridad, buscando proteger a los agentes móviles de los posibles ataques de seguridad[9]. Además, provee cierta interoperabilidad con otras plataformas.

Tracy

Se trata de una plataforma de agentes basada en Java creada conjuntamente por las Universidades de Alemania y Australia. Implementa movilidad ante un componente independiente denominado Kalong. El Kalong se caracteriza por implementar migración débil (envío del agente en forma fragmentada, bajo demanda). Últimamente ha sido integrado como un servicio de JADE para permitir migraciones inter-plataformas. La desventaja que presenta el Kalong es que está hecho bajo licencia comercial, y que tiene una comunidad de usuarios reducida que no siguen los estándares FIPA.

[8] http://jade.csel.it/.

[9] http://www.semoa.org.

Voyager

Es una plataforma que soporta las arquitecturas Cliente/Servidor, y la técnica de programación distribuida basado en agentes. Voyager es una plataforma unificada que proporciona movilidad y autonomía[10].

Voyager soporta diferentes facilidades de comunicación, permitiendo enviar mensajes Java a un agente estacionario así como a un agente en estado de movimiento, estableciendo una comunicación inter-agentes. Es de tipo comercial y no sigue los estándares FIPA.

ZEUS

Es una herramienta para construir aplicaciones multi-agente colaborativas. ZEUS define una metodología de diseño de sistemas multi-agente soportado por un entorno visual para capturar especificaciones de los agentes.

El objetivo de ZEUS[11] es facilitar el desarrollo rápido de nuevas aplicaciones multi-agente mediante la abstracción de componentes. La idea es proveer una herramienta de propósito general y personalizable que permita la creación de agentes colaborativos.

La herramienta ZEUS consiste de un conjunto de componentes, escritas en el lenguaje de programación Java, categorizada en tres grupos funcionales o librerías: una librería de *componentes* de agentes que es una herramienta de construcción de agentes, un conjunto de *utilitarios* de agentes entre los cuales podemos encontrar servidores de nombres, y los *visualizadores* de agentes.

Aquí presentamos una tabla mostrándonos a las diferentes plataformas, utilizadas en el desarrollo de agentes móviles.

Nombre, Compañía	**Tipos de Agente**	**Estándar**	**Lenguaje**	**Disponibilidad**
JADE (CSELT)	estacionario, móvil	FIPA	Java	Open-Source

[10] http://www.objectspace.com/Products/voyager.html.

[11] http://www.zeus.objectweb.org/.

FIPA-OS (Nortel Network)	estacionario	FIPA	Java	Open-source
AGLETS (IBM-Japón)	móvil	MASIF	Java	Open-Source
GRASSHOPPER (ikv ++)	móvil	MASIF/FIPA	Java	Cód. bin/sobre demanda
AJANTA (Dep. of Computer Science, Univ. Minnesota)	móvil	-	Java, Perl	Académico
ZEUS (British Telecom UK)	estacionario	FIPA	Java, KQML	Open-Source
TRACY Friedrich Schiller Univ.Alem.)	móvil	-	-	Académico
SEMOA Fraunhofer Society-Alemania)	móvil		-	Open-source
VOYAGER	móvil	-	Java	Comercial
CONCORDIA (Mitsubishi Electric Infor.)	móvil	-	Java	Comercial
D'Agents	móvil	-	TCL	Open-source
AgentDevelopment Kit (Tryllian)	móvil	FIPA	Java	Comercial/libre

Tabla 6.1 Tabla comparativa de plataformas de agentes móviles

En la tabla 6.1 se presenta un resumen de las doce plataformas móviles más importantes que se analizaron con el objetivo de elegir a uno de ellos, y el cual prometiera el desarrollo de aplicaciones aptas a las necesidades actuales, teniendo en cuenta entre otros aspectos como la seguridad, código abierto, estar en constante crecimiento de desarrollo continuo y cumplir con las especificaciones de una de las instituciones reconocidas como es FIPA.

6.7 JADE - Java Agent DEvelopment Framework

Es un middleware desarrollado por TILab para el desarrollo de aplicaciones distribuidas y sistemas multi-agentes basados en la arquitectura de comunicaciones punto a punto conforme a los estándares FIPA [Bellifemine03] (la última versión ha sido distribuida en Julio del 2007, y ya se encuentra disponible). La plataforma de agentes puede ser distribuida a través de las máquinas (el cual ni siquiera necesitan pertenecer al mismo sistema operativo) y la configuración puede ser controlada vía GUI remota.

JADE está totalmente codificado en Java que permiten construir sistemas multi-agentes robustos y escalables, basadas en los siguientes principios:

- Interoperabilidad: JADE cumple con las especificaciones FIPA, como consecuencia un agente JADE puede interoperar con otros agentes, que cumplan con el mismo estándar.
- Uniformidad y Portabilidad: JADE proporciona un conjunto de APIs que son independiente de la red y versión de Java.
- Negociación y Coordinación: JADE simplifica el desarrollo de aplicaciones que requieren de negociación y coordinación entre los agentes dentro del entorno distribuido.
- Pro-actividad: Los agentes JADE controlan sus propios hilos de ejecución y por consiguiente, pueden programarse para comenzar su ejecución de acciones o tareas sin la intervención humana.
- Aplicaciones multipartidarias: La arquitectura punto a punto es más eficaz que la arquitectura cliente servidor para el desarrollo de aplicaciones multipartidarias, ya que el servidor puede ocasionar un cuello de botella y llegar al punto de hacer fracasar al sistema entero. Los agentes JADE permiten a los clientes comunicarse con otro sin la intervención de un servidor central. Es más, el hecho que la inteligencia, la información y el control sean distribuidos, permitirá realizar aplicaciones entre los agentes, dado que cada agente puede ser capaz de realizar sólo un subconjunto de acciones de la aplicación.
- Sistema abierto: JADE es un proyecto open-source que involucra las contribuciones y colaboraciones de la comunidad de usuarios. Esto permite

a los usuarios y desarrolladores contribuir con las sugerencias y nuevos códigos que garantizan la abertura y utilidad de las APIs.

- Aplicaciones fáciles de usar y mover: Las APIs de JADE están diseñado para simplificar el manejo de comunicación y transporte de mensajes haciéndolo transparente al administrador.

JADE también ha sido integrado con JESS, una capa Java de CLIPS para explotar las capacidades de razonamiento [Vaucher03]. Actualmente está siendo utilizado por un número de compañías y grupos académicos, ambos miembros y no miembros de FIPA, como British Telecommunication, Imperial collage, Universidad de Helsinki, y muchos otros.

JADE presenta un entorno que está compuesto por varios contenedores, los cuales están distribuidos en uno o más nodos a través de la red de computadoras. El contenedor más importante que se tiene en la plataforma es el contenedor principal, el cual debe ser activado primero para que los contenedores secundarios puedan registrarse sobre la plataforma [Caire08]. Si un contenedor principal es activado en otro nodo de la red, este constituirá otra plataforma (Figura 6.4).

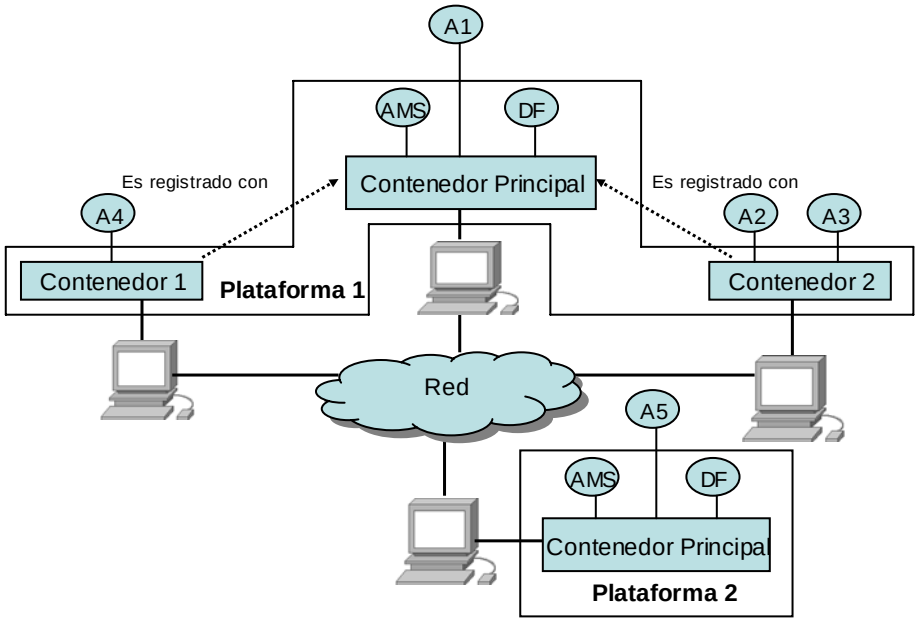


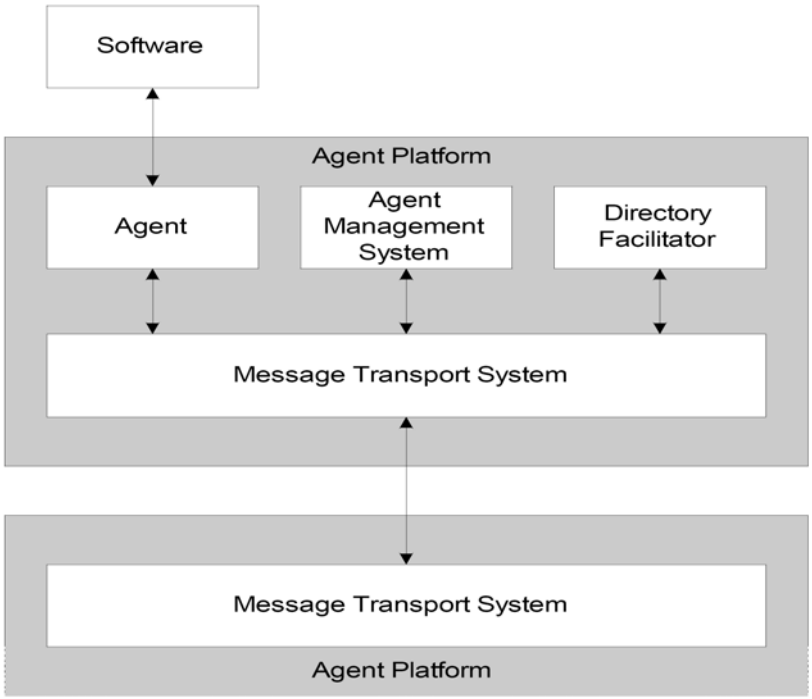
Figura 6.4 Plataformas y contenedores en entorno JADE [Caire08].

La plataforma provee de una capa homogénea para los agentes, o sea el desarrollo de las aplicaciones se hará sin importar el hardware, sistema operativo, tipo de red, o JVM. Actualmente JADE está logrando un gran impulso debido a

que es un middleware el cual simplifica el desarrollo de las aplicaciones distribuidas, compuestas por entidades autónomas que necesitan colaborar y comunicarse con otros agentes. El RMI (Invocación de Métodos Remotos) es creado por el contenedor principal, el cual será utilizado internamente por JADE para establecer la comunicación con otros host dentro de la red.

6.7.1 PLATAFORMA DE AGENTES JADE

Además de la característica de registrar a los agentes de otros contenedores, el contenedor principal se caracteriza por ser la residencia de otros tres agentes especiales (Figura 6.5), los cuales se inician automáticamente cuando se carga el contenedor principal.



6.5 Plataforma de agentes JADE definida por FIPA [FIPA023]

Sistema de Gestión de Agentes (AMS): Es el agente quien ejerce la función de supervisor y control sobre el acceso y uso de la plataforma. Sólo un AMS puede existir en una plataforma, el cual proporcionará el servicio de nombre y asegurará que cada agente en la plataforma tenga un único

nombre, manteniendo un directorio de identificador de agentes (AID) y estado del agente.

El AMS proporciona los servicio de páginas blancas y el ciclo de vida del agente.

Cada agente debe registrarse con el AMS para obtener un AID válido.

El Facilitador de Directorio (DF): Es el agente que proporciona el servicio de páginas amarillas que permite localizar a un agente por sus capacidades y no por su nombre.

El Sistema de Transporte de Mensaje: También llamado Canal de Comunicación de Agentes (ACC), es el componente de software que controla todos los intercambios de mensaje dentro de la plataforma, incluyendo mensajes a/desde plataformas remotas.

6.7.2 BEHAVIOURS

El trabajo actual que un agente debe hacer es llevado a cabo por los "behaviours", conocidos vulgarmente como comportamientos [Bellifemine07]. Un comportamiento representa una tarea que un agente puede realizar, y esta implementada como un objeto de la clase que extiende a jade.core.behaviours.Behaviour. Para hacer que un agente ejecute una tarea será suficiente añadir un behaviour al agente por medio del método addBehaviour de la clase Agent. Los comportamientos pueden ser agregados en cualquier momento desde el momento en que el agente se inicia (en el método setup()) ó dentro de otros behaviuors (Figura 6.6).

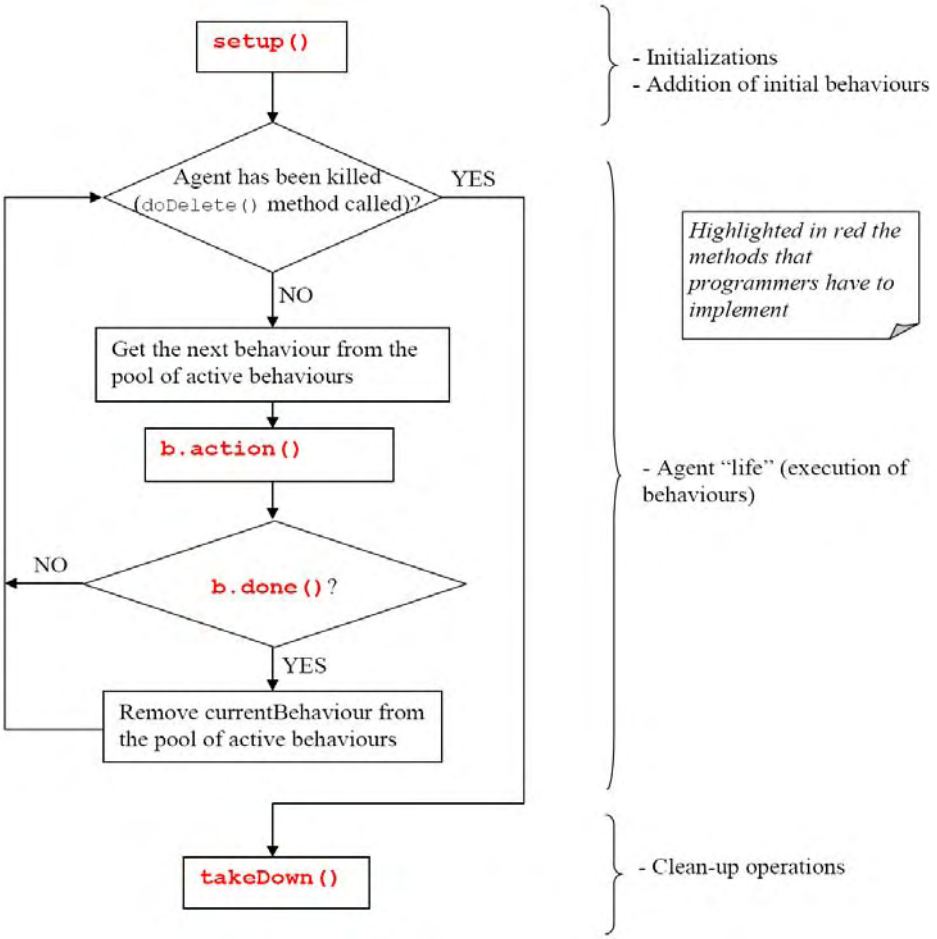


Figura 6.6 Línea de ejecución de un agente [Caire08].

Cada clase extendiéndose a un Behaviour puede implementar el método action() que es en realidad la operación que va a ser ejecutada por el comportamiento, y el método done() que retornará un valor boolean que especificará que el comportamiento ha sido o no ejecutado y de ser así, debe de ser removidos del pool de comportamientos por el agente que está llevándose a cabo.

Un agente puede ejecutar varios comportamientos concurrentemente y ser cooperativo, y es el programador quien define cuando un agente puede intercambiar de un comportamiento a otro para la ejecución de la siguiente tarea dentro de la misma aplicación.

Dentro de la notación del diagrama de clases para los comportamientos JADE, éste se inicia desde la clase básica Behaviour, una clase jerárquica

la cual está definida en el paquete jade.core.behaviour del framework JADE., como lo muestra el siguiente diagrama de estado UML (Figura 6.7)

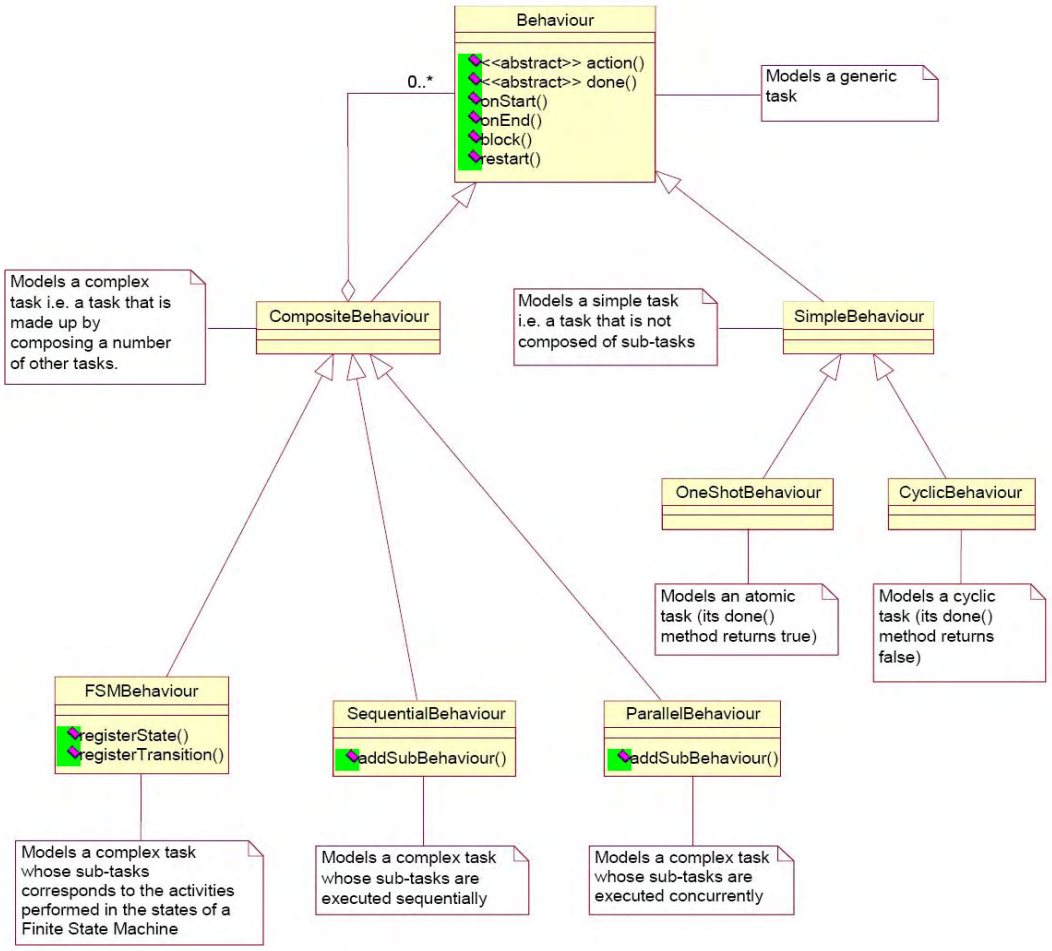


Figura 6.7 Modelo UML de los Behaviuors de JADE [Bellifemine08].

6.7.3 COMUNICACIÓN ENTRE AGENTES JADE

Una de las características más importantes que los agentes pueden proporcionar es la habilidad de comunicación. El paradigma de comunicación adoptado es el *paso de mensajes asíncronos.* Cada agente tiene como una especie de buzón (la cola de mensajes del agente) donde el *runtime JADE* apila los mensajes enviados por otros agentes (Figura 6.8). Cada vez que un mensaje sea apilado en la cola de mensajes, el

agente receptor deberá ser notificado para recoger el mensaje de la cola de mensajes y procesarla.

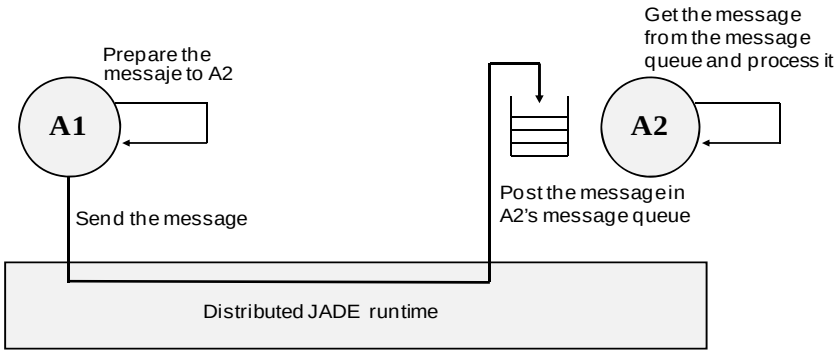


Figura 6.8 Paso de mensajes asíncronos JADE [Caire08].

6.7.3.1 EL LENGUAJE ACL

El intercambio de mensajes por los agentes JADE tiene un formato, especificado por el lenguaje ACL definida por los estándares FIPA [FIPA061] y por la interoperabilidad de los agentes. Este formato consiste de un número de campos en particular como:

Sender - que denota la identificación del remitente del mensaje, por ejemplo del nombre del agente que realiza el acto de comunicación.

Receiver - es la identificación del agente que recibirá el mensaje.

Performative – que especifica el tipo de acto comunicativo, que identificará al mensaje que se está enviando. FIPA identificó varios actos comunicativos como: ACCEPT PROPOSAL, AGREE, CANCEL, CALL FOR PROPOSAL, CONFIRM, FAILURE, PROPOSE, QUERY IF, QUERY REF, REFUSE y REQUEST.

Content - la información actual incluida dentro del mensaje (por ejemplo la acción a ser ejecutada).

Reply-with - el cual introduce una expresión para ser utilizado por el agente respondedor para identificar al mensaje.

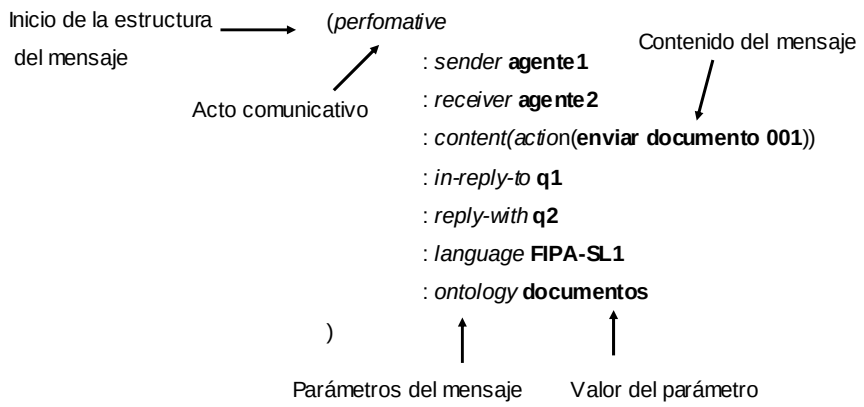
In-reply-to - hace referencia a una temprana acción por el cual el mensaje es una réplica.

Language - que denota la sintaxis utilizada para expresar el contenido.

Ontology - que describe el vocabulario que será utilizado para dar un significado a los símbolos en la expresión del contenido.

Protocol - introduce un identificador, en el cual denota al protocolo empleado por el agente remitente. Este protocolo sirve para conceder contexto adicional a la interpretación del mensaje y la comunicación entre los agentes.
Conversation-Id - introduce una expresión, el cual es usado para identificar una secuencia continua de un acto comunicativo dentro de una conversación. Una conversación puede ser utilizada por el agente para gestionar estrategia de actividades y de comunicación.

El siguiente ejemplo muestra la descripción de un mensaje usando los estándares FIPA-ACL.



La figura 6.9 representa el modelo de comunicación definido por FIPA y la relación entre estos componentes. Una de las principales ventajas de los estándares FIPA es su status “Standard”, definida y aceptada por la comunidad de agentes [Bellifemine03].

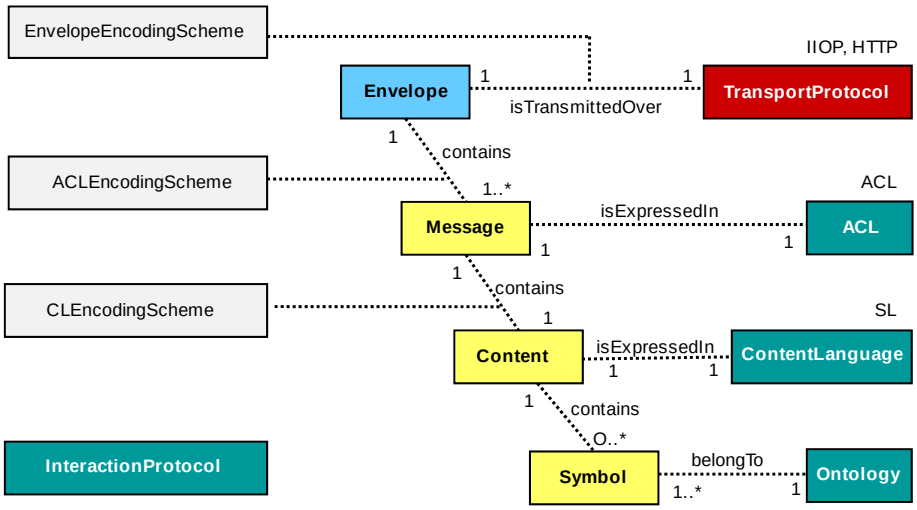


Figura 6.9 Componentes del modelo de comunicación según el estándar FIPA [Bellifemine03].

El modelo de comunicación entre agentes como se mencionó es asíncrono, los agentes envían/reciben objetos java que representan mensajes ACL por medio del lenguaje SL0 definido por FIPA [FIPA008] a través de colas de mensajes, además de añadirse los protocolos de interface como MTP (Protocolo de Transporte de Mensajes) y el IMTP (Protocolo de Transporte de Mensajes Internos).

Los agentes dentro de un mismo container pueden pasarse los mensajes mediante paso de eventos "event passing", mientras que los agente ubicados en contenedores diferentes se pasan los mensajes mediante RMI, y los agente ubicados en diferentes plataformas vía IIOP, HTTTP, a través del canal del ACC (Canal de Comunicación de agentes).

6.7.4 PROTOCOLOS DE INTERACCIÓN

FIPA posee un conjunto de protocolos de interacción estándar, los cuales pueden ser utilizados como plantillas para establecer conversación entre los agentes. Para cada conversación entre agentes, JADE distingue roles para cada uno de los agentes que participan en una conversación. El rol *Initiator* (el agente iniciando la conversación) y el rol *Responder* (el agente enganchado a la conversación después de haber sido contactado por el otro agente). JADE proporciona Behaviour (Clases) ya construidas para ambos roles de conversación siguiendo a los diferentes protocolos de interacción FIPA. Estas

clases pueden ser encontradas en el paquete jade.proto, el cual ofrece un conjunto de métodos callback (retorno de llamadas) para manipular el estado de los protocolos dentro de los API's homogéneos.

Todo comportamiento o rol *Initiator* será removido desde la cola de tareas del agente, tan pronto llegan alcanzar el estado final del protocolo de Interacción, esto permitirá el re-uso de objetos Java representados para este comportamiento sin tener que crear otra vez nuevos objetos.

Todo comportamientos o rol *Respondedor*, en cambio son cíclicos y serán reprogramados tan pronto ellos lleguen a alcanzar cualquier estado final del protocolo de interacción.

JADE en sus últimas versiones proporciona una serie y efectiva implementación de protocolos de interacción como FIPA-Request-like, FIPA-Request, FIPA-Query, FIPA-Request-When, FIPA-Recruiting, FIPA-Brokering, y así sucesivamente.

El protocolo utilizado dentro de esta investigación es el Protocolo FIPA-Request, cuya representación UML está representada en la siguiente figura 6.10 y sigue el estándar FIPA [FIPA026].

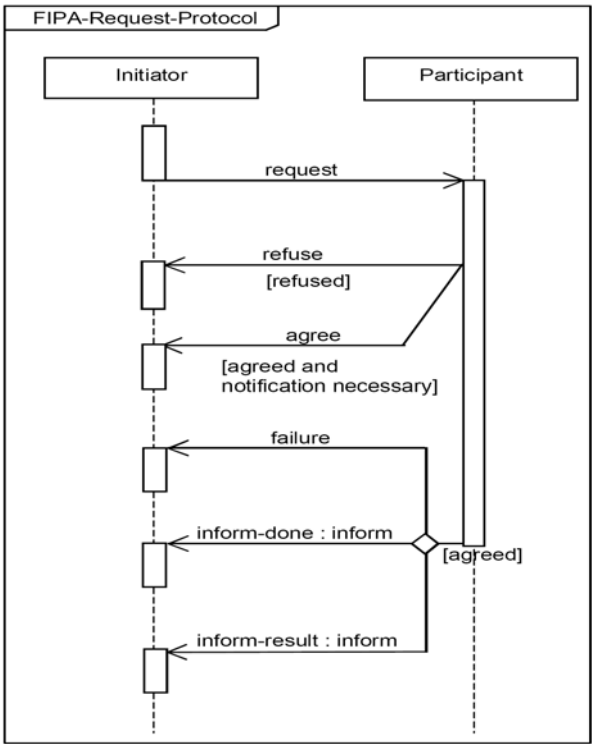


Figura 6.10 Especificación del protocolo de Interacción fipa-request [FIPA026]

El aspecto fundamental de los mensajes FIPA ACL es que un mensaje representa una acto de comunicación, que es una de las acciones que un agente puede ejecutar. El estándar FIPA especifica que para cada acto de comunicación deben de haber precondiciones de posibilidad (la condición en la cual necesita que sean verdaderas antes de una agente ejecute la acción, por ejemplo antes de que se envie un mensaje). Para así establecer el Efecto Racional (ER), quiere decir la razón por la cual se envió el mensaje.

En el protocolo de interacción fipa-request, el iniciador envía un mensaje (ejecutar el acto de comunicación). El respondedor puede entonces replicar enviando un refuse para permitir el efecto racional del acto de comunicación, o también un mensaje agree para realizar los convenios de comunicación. El respondedor ejecuta la acción y, finalmente responder con un inform del resultado de la acción (que la acción ha sido hecha ó realizada) ó con un failure si ha ocurrido un error.

6.7.5 CICLO DE VIDA DE UN AGENTE JADE

La clase Agent representa una clase base común para los usuarios que definen agentes. Por lo tanto, desde el punto de vista del programador, un agente JADE es simplemente una instancia de la clase Java que extiende a la clase Agent, lo cual simplifica el lineamiento de herencia, características para llevar a cabo las interacciones básicas con la plataforma de agentes (registración, configuración, gestión remota, etc).

El modelo computacional de un agente es la multitarea donde las tareas (behaviuors) son ejecutadas concurrentemente por el agente, entonces podemos especificar que, un agente JADE puede estar en uno de los diferentes estados del ciclo de vida de la plataforma de agentes (Figura 6.11).

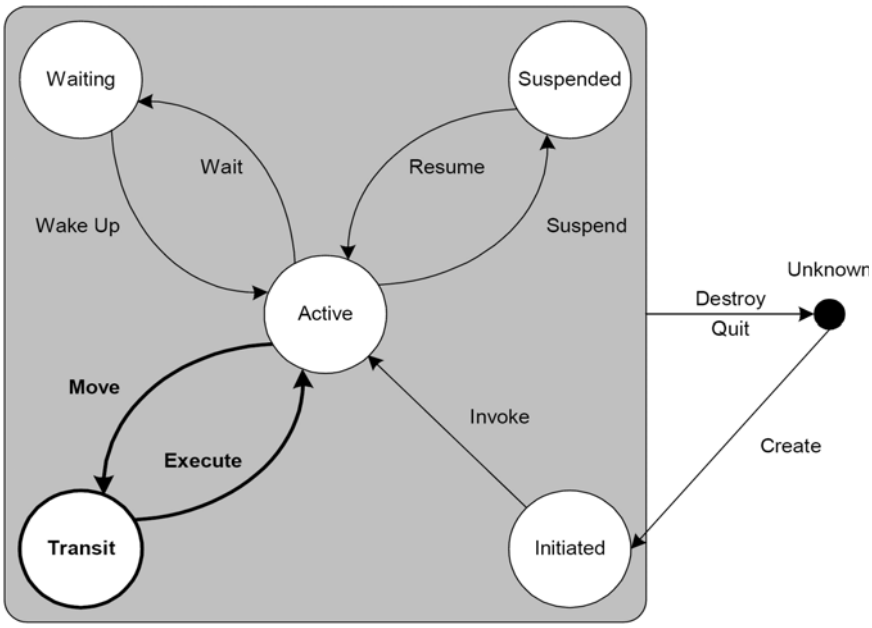


Figura 6.11 Ciclo de vida de un agente JADE [FIPA023].

INICIADO: El objeto agente es construido, pero no es registrado aún con el AMS, tampoco tiene un nombre y una dirección y no puede comunicarse con otro agente.
ACTIVO: El objeto agente es registrado con el AMS, tiene un nombre regular y dirección y puede acceder a las diferentes características de JADE.
SUSPENDIDO: El objeto agente está actualmente parado. El hilo interno está suspendido y ningún comportamiento de agente será ejecutado.
ESPERA: El objeto agente está bloqueado, esperando por algún evento. El hilo interno está durmiendo sobre un monitor Java y se despertará cuando encuentre una condición (típicamente cuando arribe un mensaje).
ELIMINADO: El objeto está definitivamente muerto. El hilo interno ha terminado su ejecución y el agente no es más registrado con el AMS.
TRÁNSITO: Un agente móvil entra a éste estado mientras esta migrando a una nueva localización. El sistema mantiene el buffer de mensajes que será enviado a su nueva localización.

6.7.6 SOPORTE DE MOVILIDAD EN JADE

De acuerdo con (Bellifemine et al., 2005a) JADE se restringe al soporte de movilidad entre contenedores de una misma plataforma. Se pueden desarrollar agentes móviles con capacidades de migrar o clonarse ellos mismos a lo largo de

múltiples nodos de una red, lo que involucra una transición de estados en el ciclo de vida de un agente. La movilidad puede ser iniciada por el propio agente o por el AMS [FIPA087]. Un agente móvil necesita conocer su localización para decidir cuando y donde desplazarse, por lo que JADE ha desarrollado una ontología (jade.mobility.ontology) con conceptos y acciones necesarias (jade.domain.mobility).

JADE proporciona un API para la movilidad de la clase agent, el cual dispone del método: doMove(), que recibe como parámetro un jade.core.Location, que representa el destino o localización donde migrará el agente, es decir las aplicaciones de agentes no pueden crear sus propias localizaciones, sino que tienen que preguntar al AMS la lista de localizaciones disponibles para poder trasladarse y elegir una de ellas. Un agente también le puede preguntar al AMS la localización donde se encuentra un agente determinado.

Mover un agente implica la transmisión de, al menos el código y los datos, y posiblemente el estado (proceso de serialización y deserialización), algunos de los recursos usados por el agente móvil se moverán con él, mientras que otros serán desconectados antes de salir y reconectados en el destino.

JADE proporciona una serie de métodos para la gestión de recursos dentro de la clase agent: beforeMove(), afterMove(), beforeClone() y afterClone().

El AMS para llevar a cabo la movilidad necesita de dos acciones: la ontología (jade.mobility.ontology) y el lenguaje de contenido (fipa - SL0).

CAPITULO 7 Arquitectura del Agente Móvil SNMP, Modelamiento, Implementación y Análisis en el Entorno Distribuido

En este capítulo, se describirá el procedimiento llevado a cabo para el desarrollo del agente móvil con capacidades SNMP y su arquitectura de interacción en la red. Luego se detallará su funcionamiento, describiendo a cada uno de los elementos que lo conforman dentro del proceso de gestión, así como el modelamiento de clases ó behaviour que lo conforman.

Posteriormente se realizará el análisis cuantitativo de dicho modelo, para finalmente llegar a su implementación en la estación de gestión y por consiguiente la obtención de los resultados de dicha operación. Este sistema de agente móvil ha sido desarrollado bajo la plataforma JADE.

7.1 CREACIÓN DEL AGENTE MÓVIL SNMP

El agente móvil no es más que un programa desarrollado en Java, cuyo diseño está orientado a objetos, el cual modelará el software en términos similares a las que utilizan las personas para describir objetos del mundo real, dentro de la cual se aprovechan las relación entre las clases [Deitel04].

El procedimiento ó ruta seguida para el desarrollo del agente móvil con características de gestión SNMP se denomina “AgenteMobilSNMP - AMSNMP”, el cual se muestra en el siguiente diagrama de flujo.

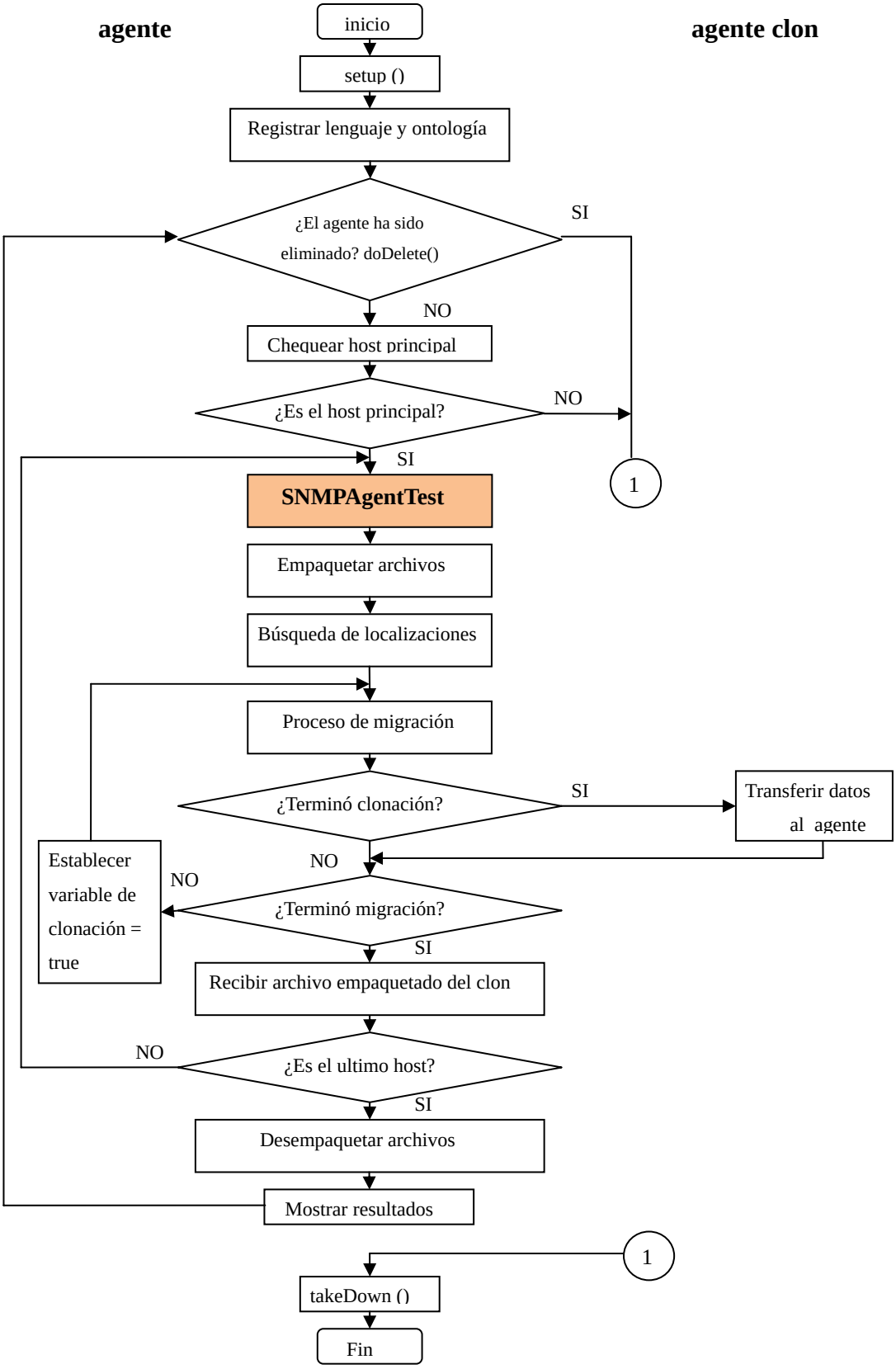


Figura 7.1 Diagrama de flujo de desarrollo del agente móvil "AgenteMobil SNMP" [ELLS].

7.2 ARQUITECTURA

La arquitectura del sistema de gestión desarrollado, en donde el objetivo principal es generar un objeto agente móvil con capacidades de gestión SNMP (AMSNMP), se representa de la siguiente manera (Figura 7.2).

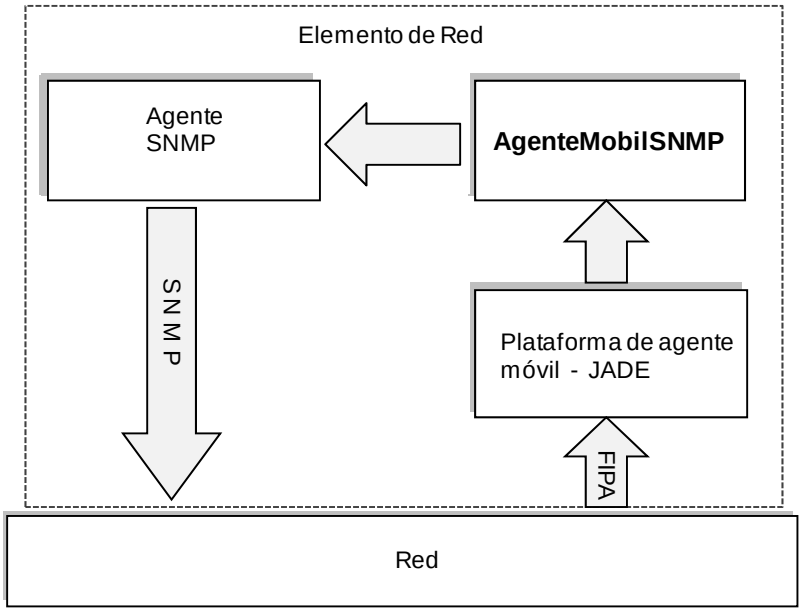


Figura 7.2 Arquitectura del agente móvil SNMP [ELLS].

Donde:

Plataforma de agente móvil: Nos permitirá inter-operar con los diferentes elementos que componen la red, y dentro del cual se ha desarrollado al agente móvil.

AgenteMobilSNMP: Es el corazón o núcleo del sistema de gestión que realiza la operación de gestión.

Agente SNMP: Es el agente estático de gestión de cada elemento que componen la red y de quién se obtendrán los valores de las variables u objetos MIB, para las operaciones de gestión.

7.3 FUNCIONAMIENTO

La figura 7.3 muestra el funcionamiento del agente móvil SNMP, quien se inicia en la estación de gestión, para luego trasladarse a los diferentes nodos de la red, para realizar las tareas (behaviuors) encomendadas por el gestor de agente móvil.

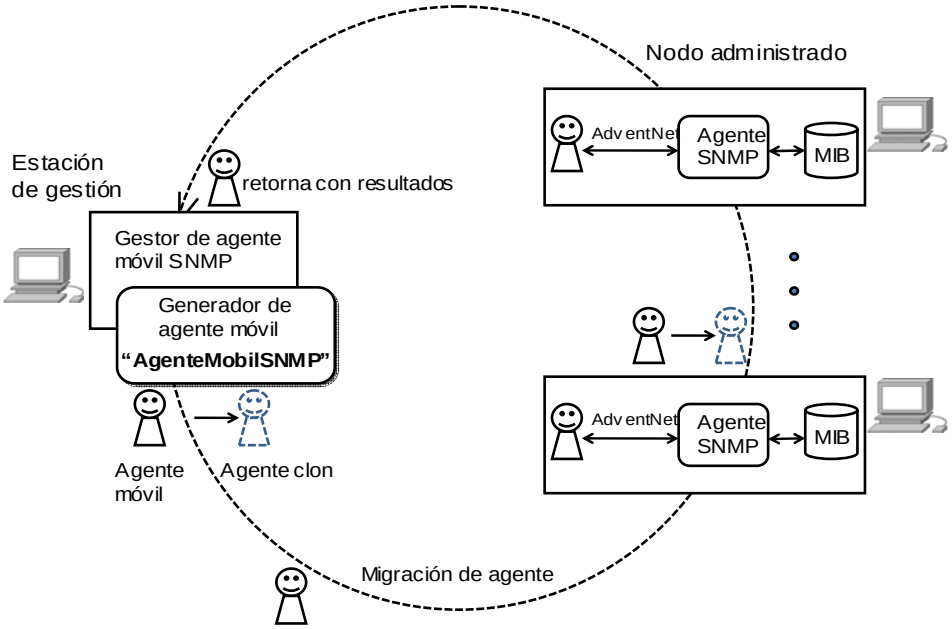


Figura 7.3 Funcionamiento del agente móvil "AgenteMobilSNMP" [ELLS].

Gestor de agente móvil SNMP: Es el programa desarrollado en Java, que se encargará de la creación del agente móvil AMSNMP. Proporcionará las clases (tareas o behaviours), siendo una de ellas las operaciones de gestión a cada una de las estaciones que visite, para luego empaquetarlo en un archivo de texto.

El agente: Es el agente móvil original, cuya función la podemos detallar de la siguiente manera: realizar las operaciones de gestión → empaquetar resultados → realizar el proceso de clonación → realizar migración al siguiente nodo.

El agente clon: Se encargará de interactuar con el agente original a través del protocolo de interacción fipa-request de FIPA, pasándole los resultados a través de objetos de tipo mensajes empaquetados al agente original. Después de la transferencia del mensaje, este agente clon se auto-elimina a través de otro behaviour.

El agente SNMP: Se encargará de interactuar con los objetos del MIB, para la obtención de los objetos utilizados en la evaluación de los parámetros de performance.

El MIB: Proporcionará los valores de cada uno de los objetos.

De manera resumida podemos explicar, que se crea un agente móvil, que realizará operaciones de gestión a cada nodo que encuentre, que a su vez empaquetará los

resultados, luego éste migrará al siguiente punto o nodo de la red, pero antes de trasladarse, se clona, para poder establecer la comunicación entre éste y su clon, donde el clon debe pasarle los resultados calculados y empaquetados por el agente original, esto es a través de mensajes ACL. La creación del agente clon dentro del desarrollo de este sistema de gestión es por el impacto significante en la eficiencia en los sistemas de monitoreo, sobre todo en el despliegue de los MA y tráfico [Liotta99], el cual son factores críticos en sistemas de gran escala y particularmente beneficioso en el caso de redes jerárquicas.

El código Java del AgenteMobilSNMP - AMSNMP se muestra en el anexo C.

7.4 MODELAMIENTO DE CLASES

A. Diagrama de Clases

Dentro de las clases y métodos utilizados para el desarrollo del agente móvil AMSNMP podemos mencionar (Figura 7.4):

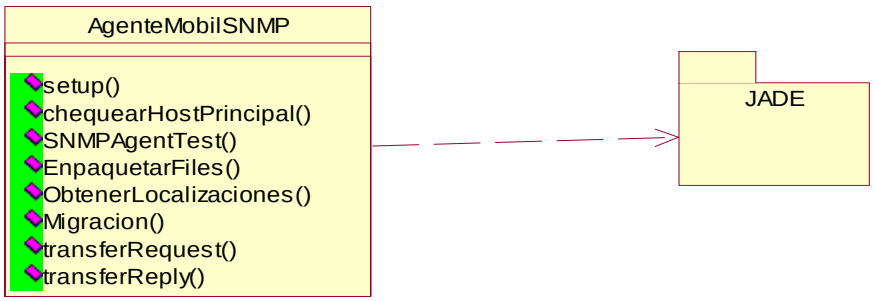


Figura 7.4 Estructura de clases del agente móvil AMSNMP [ELLS].

setup(): Método principal del agente AMSNMP, en donde se encuentran todos los behaviours o tareas a realizar por el agente móvil.

ChequearHostPrincipal(): Comportamiento que chequea si la estación es la estación de gestión principal (NMS), el cual será utilizado para realizar las comparaciones con las demás localizaciones a gestionar.

SNMPAgentTest(): Comportamiento que realiza la operación de gestión de red en cada estación que visite.

EmpaquetarFiles(): Comportamiento que realiza el empaquetamiento de los datos obtenidos de la estación gestionada.

ObtenerLocalizaciones(): Comportamiento que solicita al AMS las posibles localizaciones donde migrar. Realiza el proceso de clonación del agente original.

Migracion(): Comportamiento que realiza el proceso de migración del agente al host destino o localización dada por el AMS.

TransferRequest(): Comportamiento de interacción del protocolo fipa-request, quien recibirá los datos enviados por el agente clon, y guardarlos en un archivo de texto.

TransferReply(): Comportamiento de interacción, por el cual el agente clon enviará los datos en forma de mensaje al agente original.

Todas estas clases se encuentran dentro del paquete denominado “tesis”

B. Diagrama de Secuencias

Describe la secuencia de tareas realizadas por el agente móvil para la realización de la operación de gestión. Luego realiza el proceso de clonamiento del mismo, quién es utilizado como el agente que enviará los resultados cuando éste (agente original) viaje a la siguiente localización o destino.

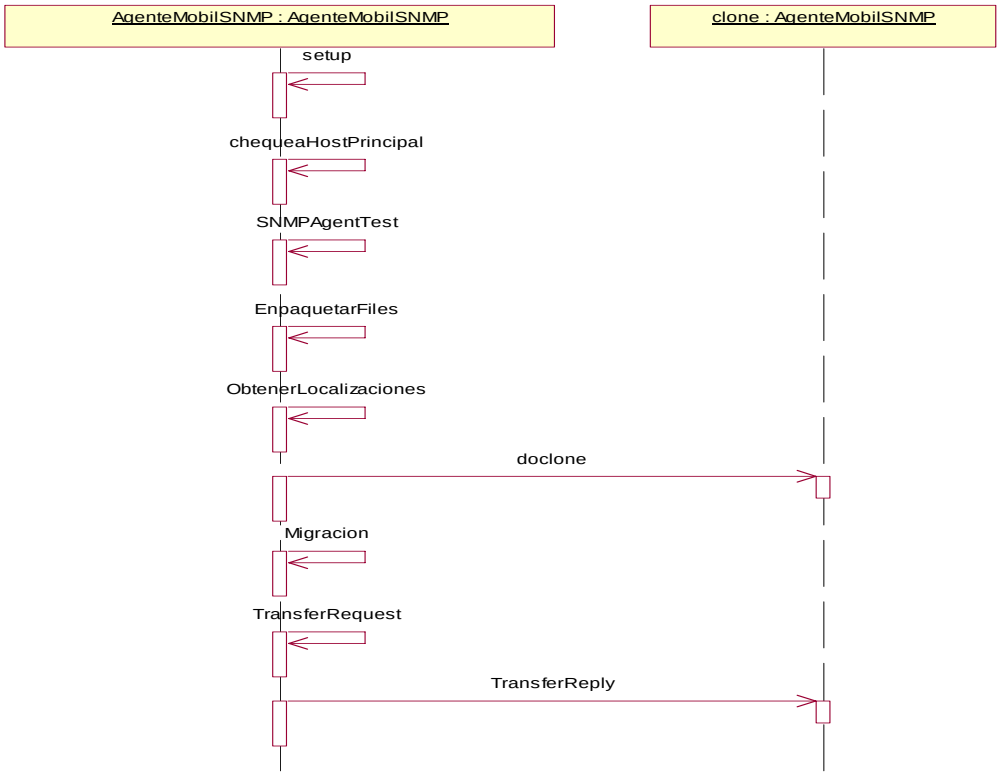


Figura 7.5 Intercambio de mensajes entre el agente original y su clon [ELLS].

C. Diagrama de Actividades

El diagrama de actividades de la figura 7.6 muestra la línea de ejecución de cada una de las tareas encomendadas por el agente móvil AMSNMP.

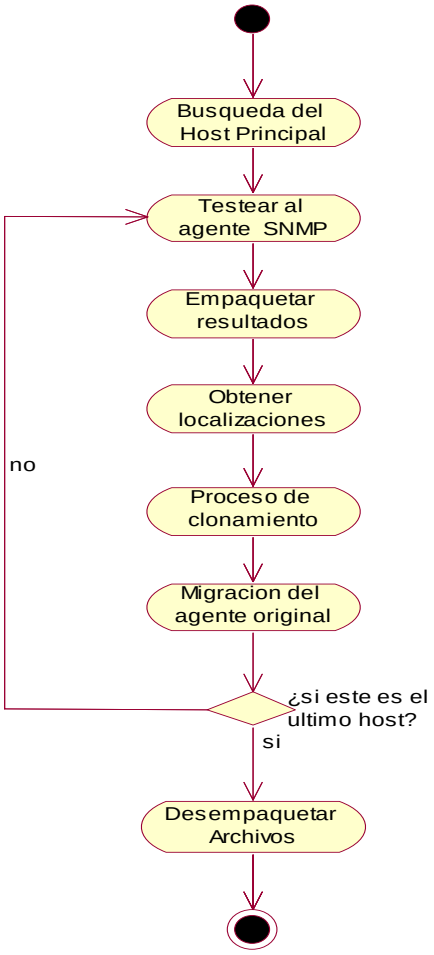


Figura 7.6 Estados que sigue el agente móvil AMSNMP [ELLS].

7.5 ANÁLISIS CUANTITATIVO DEL TRÁFICO DE GESTIÓN

En esta sección ilustramos el tráfico de gestión originado dentro del entorno distribuido, mediante la aplicación del agente móvil AMSNMP.

Este agente móvil (constituido por el código, estado y las variables para la recolección de datos) se crea en la estación de gestión (NMS), que luego de realizar la operación de gestión en la misma se trasladará (código y los datos) a la siguiente estación a gestionar, retornando al final con los resultados de todas estaciones gestionadas.

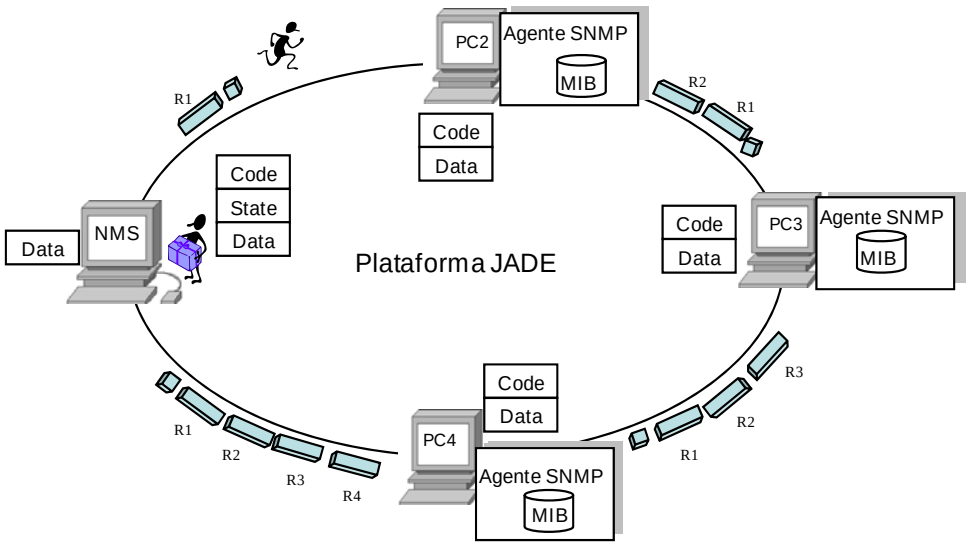


Figura 7.7 Tráfico de gestión en el modelo distribuido por el agente móvil AMSNMP [ELLS].

De la figura 7.7, se describe lo siguiente:

- El tráfico a través de la NMS es solo cuando el agente móvil es inyectado dentro de la red, y cuando de regreso reporta los resultados.
- El agente móvil original realiza la operación de gestión en un determinado nodo antes de viajar al siguiente nodo. Luego los resultados serán enviados por el agente clon cuando éste (agente original) arribe al nodo destino.
- Se considera a la NMS como el nodo 1 para el desarrollo numérico de la ecuación.

Luego el tráfico generado a través de los enlaces, cuando los datos y el agente arriben a cada PC estará dada por:

PC1: INICIO DE PARTIDA DEL AGENTE MÓVIL AMSNMP

PC2: $T_{MA1\rightarrow 2} = \eta_{MA} C_{MA}$ bytes + $\acute{\eta}_{MA}(1-\sigma)R_1$ bytes (1)

PC3: $T_{MA2\rightarrow 3} = \eta_{MA} C_{MA}$ bytes + $\acute{\eta}_{MA}(1-\sigma)[R_1+R_2]$ bytes (2)

PC4: $T_{MA3\rightarrow 4} = \eta_{MA} C_{MA}$ bytes + $\acute{\eta}_{MA}(1-\sigma)[R_1+R_2+R_3]$ bytes (3)

PC1: LLEGADA DEL AGENTE MÓVIL AMSNMP

$T_{MA4\rightarrow 1} = \eta_{MA} C_{MA}$ bytes + $\acute{\eta}_{MA}(1-\sigma)[R_1+R_2+R_3+R_4]$ bytes (4)

El ancho del agente móvil de entrada a la red es igual al ancho de salida.

$$T_{MA,NMS} = 2\eta_{MA}C_{MA} + \acute{\eta}_{MA}(1 - \sigma)R1 + \acute{\eta}_{MA}\sum_{n=1}^{4}(1 - \sigma)Rn \quad (5)$$

Representación general:

$$T_{MA,NMS} = 2\eta_{MA}C_{MA} + \acute{\eta}_{MA}(1 - \sigma)R1 + \acute{\eta}_{MA}\sum_{n=1}^{N}(1 - \sigma)Rn \quad (6)$$

Ecuación que representa el tráfico través de la NMS, expresado en bytes.

Donde:

C_{MA}: Ancho del código del MA en la entrada y salida de la red, incluyendo las instrucciones SNMP.

R_n: Resultados recogidos del nodo anterior gestionado.

$\acute{\eta}_{MA}$: Función de cabecera de réplica del resultado del nodo n.

η_{MA}: Función de cabecera del MA.

N: Número de nodos gestionados, incluyendo a la NMS.

σ: Tasa de compresión de datos , que varía de $0 \leq \sigma \leq 1$.

Si se quiere calcular el tráfico total en la red, solo será necesario sumar todos los tráficos generados en los enlaces:

$$T_{MAtotal} = T_{MA1\rightarrow2} + T_{MA2\rightarrow3} + T_{MA3\rightarrow4} + T_{MA4\rightarrow1} \quad (7)$$

7.6 ANÁLISIS CUANTITATIVO DEL TIEMPO DE GESTIÓN

En esta sección se describe el tiempo utilizado por el agente móvil AMSNMP en realizar toda la operación de gestión. Se hace necesario de los parámetros throughput (rendimiento) y al retardo (delay) ocasionado a través de cada uno de los enlaces.

El costo de tiempo que le llevará al agente móvil realizar el proceso de gestión está dado por:

$$Kt_{MA} = \theta T_{MA,total} \quad (8)$$

Donde:

$T_{MA,total}$: Es una generalización del tráfico de la ecuación (7), expresada en bytes.

θ: Valor de coeficiente de costo ($\lambda/T + 1/\tau$) de un determinado enlace, cuyo valor será determinado por el gestor de acuerdo a la noción del costo asociado al enlace.

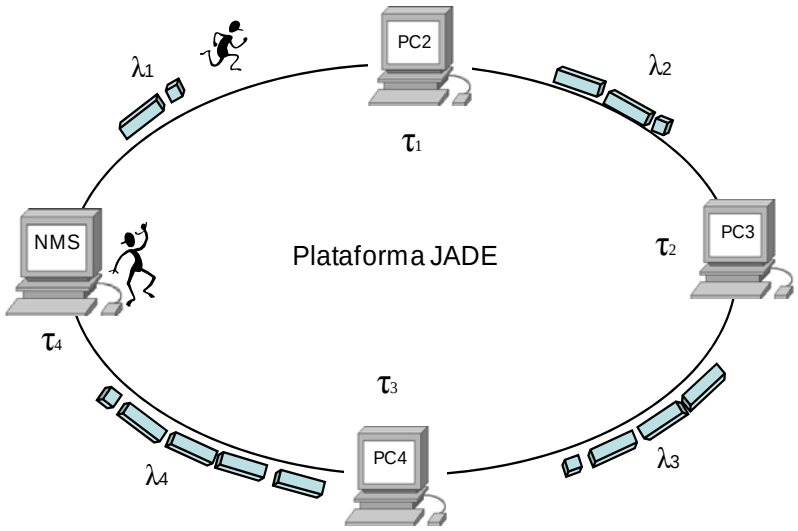


Figura 7.8 Tiempos de gestión en el modelo distribuido por el agente móvil AMSNMP [ELLS].

De la figura 7.8, los tiempos realizados en cada uno de los nodos gestionados, a la llegada de los datos después de haber realizado las operaciones de gestión estará dado por:

PC1: INICIO DE PARTIDA DEL AGENTE MÓVIL AMSNMP

PC2:

$$Kt_{MA1\rightarrow 2} = \lambda_{MA1} + \frac{\eta_{MA} C_{MA}}{\tau_{MA1}} + \lambda_1 + \frac{\acute{\eta}_{MA}(1-\sigma)R_1}{\tau_1} \quad (9)$$

PC3:

$$Kt_{MA2\rightarrow 3} = \lambda_{MA2} + \frac{\eta_{MA} C_{MA}}{\tau_{MA2}} + \lambda_2 + \frac{\acute{\eta}_{MA}(1-\sigma)[R_1 + R_2]}{\tau_2} \quad (10)$$

PC4:

$$Kt_{MA3\rightarrow 4} = \lambda_{MA3} + \frac{\eta_{MA} C_{MA}}{\tau_{MA3}} + \lambda_3 + \frac{\acute{\eta}_{MA}(1-\sigma)[R_1 + R_2 + R_3]}{\tau_3} \quad (11)$$

C1: LLEGADA DEL AGENTE MÓVIL AMSNMP

$$Kt_{MA4\rightarrow 1} = \lambda_{MA4} + \frac{\eta_{MA} C_{MA}}{\tau_{MA4}} + \lambda_4 + \frac{\acute{\eta}_{MA}(1-\sigma)[R_1 + R_2 + R_3 + R_4]}{\tau_4} \quad (12)$$

Luego, el tiempo total, estará dado por:

$$Kt_{MAtotal} = \sum_{n=1}^{4} \lambda_{MAn} + \sum_{n=1}^{4} \frac{\eta_{MA} C_{MA}}{\tau_{MAn}} + \sum_{n=1}^{4} \lambda_n + \sum_{n=1}^{4} \frac{\acute{\eta}_{MA} S_{MA,n}}{\tau_n} + \sum_{n=1}^{4} \frac{\acute{\eta}_{MA}(1-\sigma) R_n}{\tau_n} \quad (13)$$

Representación General:

$$Kt_{MAtotal} = \sum_{n=1}^{N} \lambda_{MAn} + \sum_{n=1}^{N} \frac{\eta_{MA} C_{MA}}{\tau_{MAn}} + \sum_{n=1}^{N} \lambda_n + \sum_{n=1}^{N} \frac{\acute{\eta}_{MA} S_{MA,n}}{\tau_n} + \sum_{n=1}^{N} \frac{\acute{\eta}_{MA}(1-\sigma) R_n}{\tau_n} \quad (14)$$

Donde:

$$S_{MA,n} = \begin{cases} 0 & \text{Si } n = 1 \\ \sum_{m=1}^{n-1} (1-\sigma) R_m & \text{Si } n > 1 \end{cases}$$

$S_{MA,n}$: Datos acarreados a través de los enlaces de los nodos gestionados.

N: Número de nodos gestionados, incluyendo a la NMS.

Rn: Resultados acarreados.

λ_{MAn} : Retardo de transmisión del agente móvil, en el enlace n.

λ_n : Retardo de transmisión de resultado en el enlace n.

τ_{MAn}: Throughput al arribar el agente móvil al nodo n.

τ_n: Throughput al arribar el resultado al nodo n.

C_{MA}: Ancho del código móvil, incluyendo las instrucciones SNMP

η_{MA}: Función de cabecera del MA

$\acute{\eta}_{MA}$: Función de cabecera de las réplicas (resultados obtenidos)

Estableciendo las unidades de la expresión, en donde cada uno de los términos están expresados en bytes (1 byte = 8 bits) tendremos:

$$Kt_{MAtotal} = 8\sum_{n=1}^{N}\lambda_{MAn}\ seg + 8\sum_{n=1}^{N}\frac{\eta_{MA}C_{MA}}{\tau_{MAn}}\ seg + 8\sum_{n=1}^{N}\lambda_{n}\ seg + 8\sum_{n=1}^{N}\frac{\acute{\eta}_{MA}S_{MA,n}}{\tau_{n}}\ seg + 8\sum_{n=1}^{N}\frac{\acute{\eta}_{MA}(1-\sigma)R_{n}}{\tau_{n}}\ seg \quad (15)$$

7.7 IMPLEMENTACIÓN

Las herramientas utilizadas para el desarrollo del agente móvil AMSNMP, son las mismas que se utilizaron para la aplicación de gestión, con la única agregación del paquete JADE, el cual contiene las clases, tanto para el desarrollo del agente móvil, como de los behaviours o comportamientos que utilizará para la operación de gestión.

7.7.1 PROCEDIMIENTO

Se sigue el mismo procedimiento del capítulo 5, con la única excepción que hemos añadido el middleware JADE, para establecer la plataforma, dentro de la cual se llevará todo el proceso de gestión.

- Configurar en el classpath la localización de las clases de los paquetes JADE.
- Configurar en el path de la variable de entorno, la ubicación de las clases de Java, tanto en el contenedor principal como en los contenedores secundarios.

7.7.2 EJECUCIÓN

Para la ejecución de la aplicación, primero debemos de levantar la plataforma, y luego los contenedores que se adjuntaran a éste. Tanto la plataforma, como los

contenedores se ejecutan a través del comando de línea, dentro del símbolo del sistema proporcionado por Windows.

A. El Contenedor principal

Es la parte central de la aplicación, y se inicia ejecutándose la plataforma, mediante el siguiente comando.

C:\> java jade.Boot -gui

Donde:

jade.Boot: Es el comando que ejecuta la iniciación de la plataforma JADE.

-gui: Es la interfaz gráfica, en el que se ejecutará la aplicación.

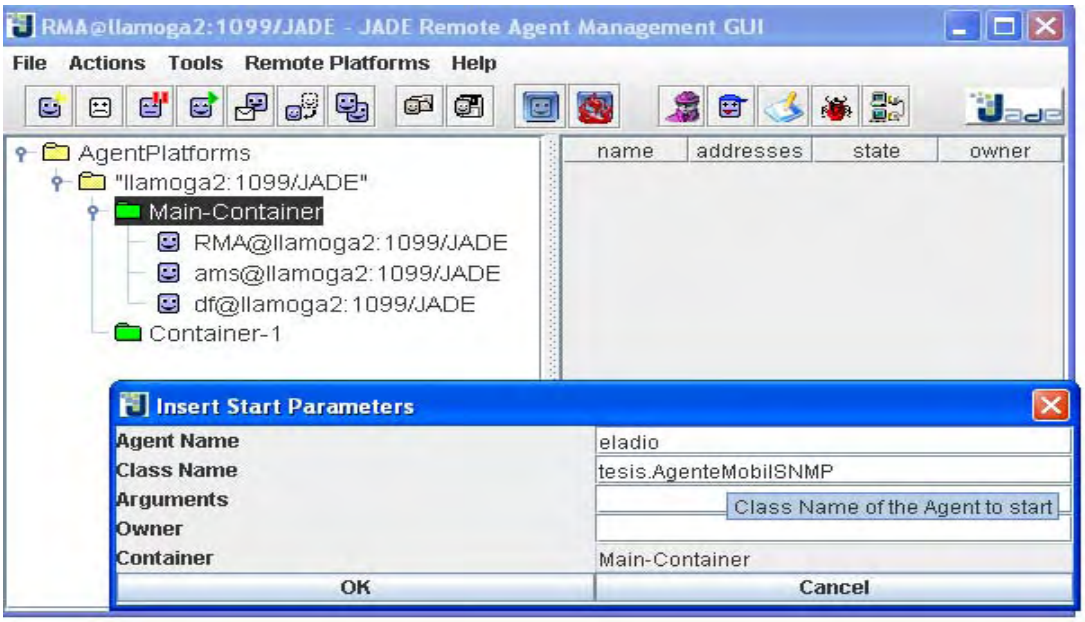


Figura 7.9 Interfaz gráfica de monitorización del "AgenteMobilSNMP" [ELLS].

B. Los contenedores secundarios

Los contenedores secundarios, son las estaciones a gestionar, también denominados "containers" y deben adjuntarse a la plataforma mediante el siguiente comando.

C :\> java jade.Boot -host llamoga2 -container.

C. Ejecución de la Aplicación

Una vez establecida la plataforma y los contenedores, escribir el nombre del paquete en donde se encuentra la aplicación del agente móvil SNMP, como lo muestra la figura 7.9.

7.7.3 RESULTADOS OBTENIDOS

Dentro de los resultados obtenidos se ha considerado el lenguaje de comunicación establecida durante el proceso de gestión entre el agente original y el agente clon, ya que de esto no permitirá visualizar el buen resultado del proceso. Y por último los datos obtenidos del test serán almacenados en un archivo “files”, los cuales serán utilizados para nuestra contrastación con el otro entorno de gestión (centralizado).

A. UTILIZACIÓN DE LA INTERFAZ

Los valores de medición de los objetos MIB obtenidos por el agente móvil AMSNMP, se muestran en las tablas 7.1, 7.2 y 7.3.

OBJETOS MIB	t 1		t2	
	ESTACION DE GESTIÓN (NMS)	ESTACIÓN GESTIONADA	ESTACION DE GESTIÓN (NMS)	ESTACIÓN GESTIONADA
ifInOctets	858071	4375924	965194	5252445
ifOutOctets	6081916	880042	7027129	1043616
ifSpeed	1.00E+08	1.00E+08	1.00E+08	1.00E+08
sysUpTime	4348.04	5314.29	4828.93	5794.79
ifInErrors	0	0	0	0
ifInUcastPkts	0	0	0	0
ifInNUcastPkts	8970	1157	957	1209

Tabla 7.1 Valores de objetos MIB de la NMS y una estación gestionada en el tiempo t1 y t2 en el entorno distribuido

OBJETOS MIB	t 1			t2		
	ESTACION DE GESTIÓN (NMS)	ESTACIONES GESTIONADAS		ESTACION DE GESTIÓN (NMS)	ESTACIONES GESTIONADAS	
ifInOctets	1141370	7608712	12781105	1436682	8829838	14550199
ifOutOctets	8245298	1469868	1726579	10647922	1623401	1921360
ifSpeed	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08
sysUpTime	5654.6	6220.51	10373.32	6475.92	7441.87	11194.59
ifInErrors	0	0	0	0	0	0
ifInUcastPkts	0	0	0	0	0	0
ifInNUcastPkts	1077	1318	2348	1246	1478	2517

Tabla 7.2 Valores de objetos MIB de la NMS y dos estaciones gestionadas en el tiempo t1 y t2 en el entorno distribuido

OBJETOS MIB	t 1				t2			
	ESTACION DE GESTIÓN (NMS)	ESTACIONES GESTIONADAS			ESTACION DE GESTIÓN (NMS)	ESTACIONES GESTIONADAS		
ifInOctets	1743790	10184501	16419938	29181334	2175454	11564512	17649630	30748503
ifOutOctets	13063846	1936695	2138456	41654253	16658491	2124736	2293229	42308988
ifSpeed	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08
sysUpTime	7657.76	8624.34	12377.12	14549.39	8615.03	9581.96	13334.82	15507.07
ifInErrors	0	0	0	0	0	0	0	0
ifInUcastPkts	0	0	0	0	0	0	0	0
ifInNUcastPkts	1413	1629	2673	3606	1549	1750	2802	3735

Tabla 7.3 Valores de objetos MIB de la NMS y tres estaciones gestionadas en el tiempo t1 y t2 en el entorno distribuido

La utilización de la interfaz después de cada operación de gestión entre:

1. El gestor de agente móvil y una estación gestionada
2. El gestor de agente móvil y dos estaciones gestionadas
3. El gestor de agente móvil y tres estaciones gestionadas

Se muestra en la siguiente tabla 7.4.

UTILIZACION DE LA INTERFAZ EN EL ENTORNO DISTRIBUIDO		
N° DE MAQUINAS	ESTACION DE GESTION (%)	ESTACIONES GESTIONADAS (%)
2 (NMS - 1PC's)	1.75064E-02	1.73168E-02
3 (NMS - 2PC's)	2.6279E-02	1.33891E-02
	2.6279E-02	1.91301E-02
4 (NMS - 3PC's)	3.36482E-02	1.30995E-02
	3.36482E-02	1.15649E-02
	3.36482E-02	1.85607E-02

Tabla 7.4 Tabla de la utilización de la interfaz en el entorno distribuido

Los datos obtenidos se muestran gráficamente en la figura. 7.10.

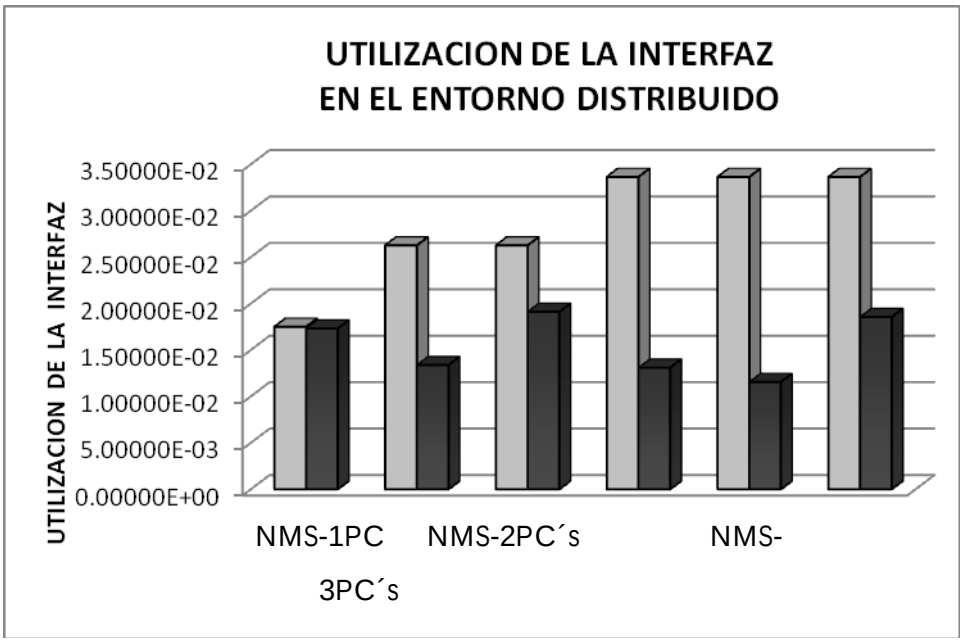


Figura 7.10 Gráfica de la utilización de la interfaz en el entorno distribuido [ELLS].

B. TIEMPOS DE GESTIÓN

La tabla 7.5 muestra los tiempos después de cada operación de gestión entre:

1. El gestor de agente móvil y una estación gestionada
2. El gestor de agente móvil y dos estaciones gestionadas
3. El gestor de agente móvil y tres estaciones gestionadas.

TIEMPOS EN EL ENTORNO DISTRIBUIDO	
N° MAQUINAS (PC's)	TIEMPO (seg)
2 (NMS - 1PC's)	3
3 (NMS - 2PC's)	6
4 (NMS - 3PC's)	12

Tabla 7.5 Tabla de los tiempos de respuesta en el entorno distribuido

La figura 7.11 muestra los datos de los tiempos obtenidos, gráficamente.

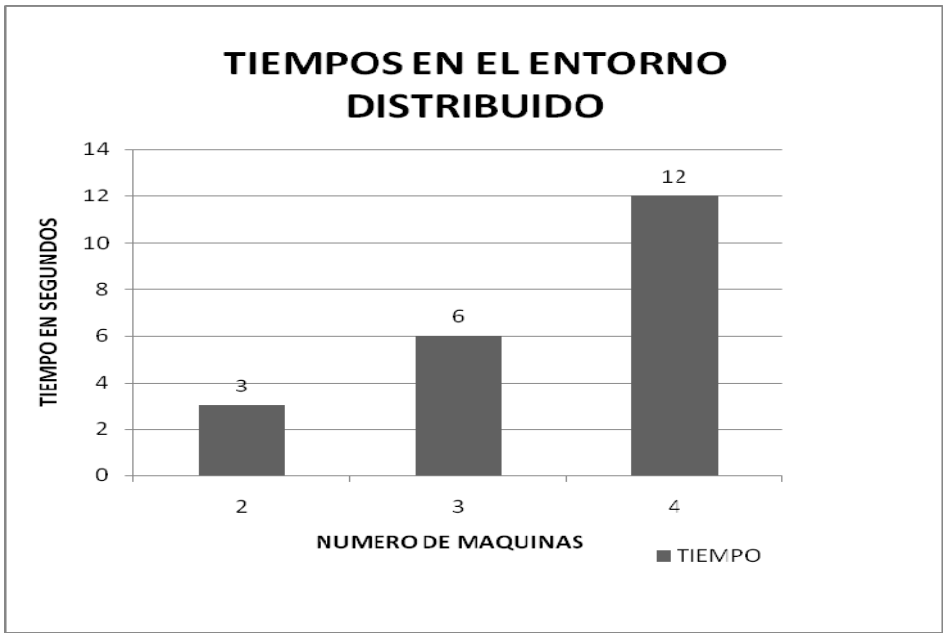


Figura 7.11 Gráfica de tiempos de respuesta en el entorno distribuido [ELLS].

- Del gráfico de la utilización de la interfaz, se hace una comparación entre la estación de gestión con cada una de las entidades gestionadas.
- Por otro lado para la evaluación de la tasa de error, no se pudieron graficar, esto es debido, a que la evaluación da como resultado cero, no obteniéndose resultado alguno.
- El porcentaje de utilización de la interfaz en la NMS y en las entidades gestionadas disminuye de acuerdo a la cantidad de estaciones añadidas.

- El tiempo de gestión va incrementándose con la cantidad de estaciones gestionadas.

C. EL ACTO DE COMUNICACIÓN

a. En el Contenedor Principal

Es la comunicación establecida entre el agente original y su clon. Visualiza si se ha llevado correctamente el proceso de gestión, así como la obtención del archivo con el/los datos después de cada operación de gestión (Figura 7.12).

Dentro del contenedor principal podemos observar:

- Las performativas: REQUEST, AGREE, INFORM, establecidas entre el agente original (eladio) y el agente clon (eladio clone) durante la operación de gestión.

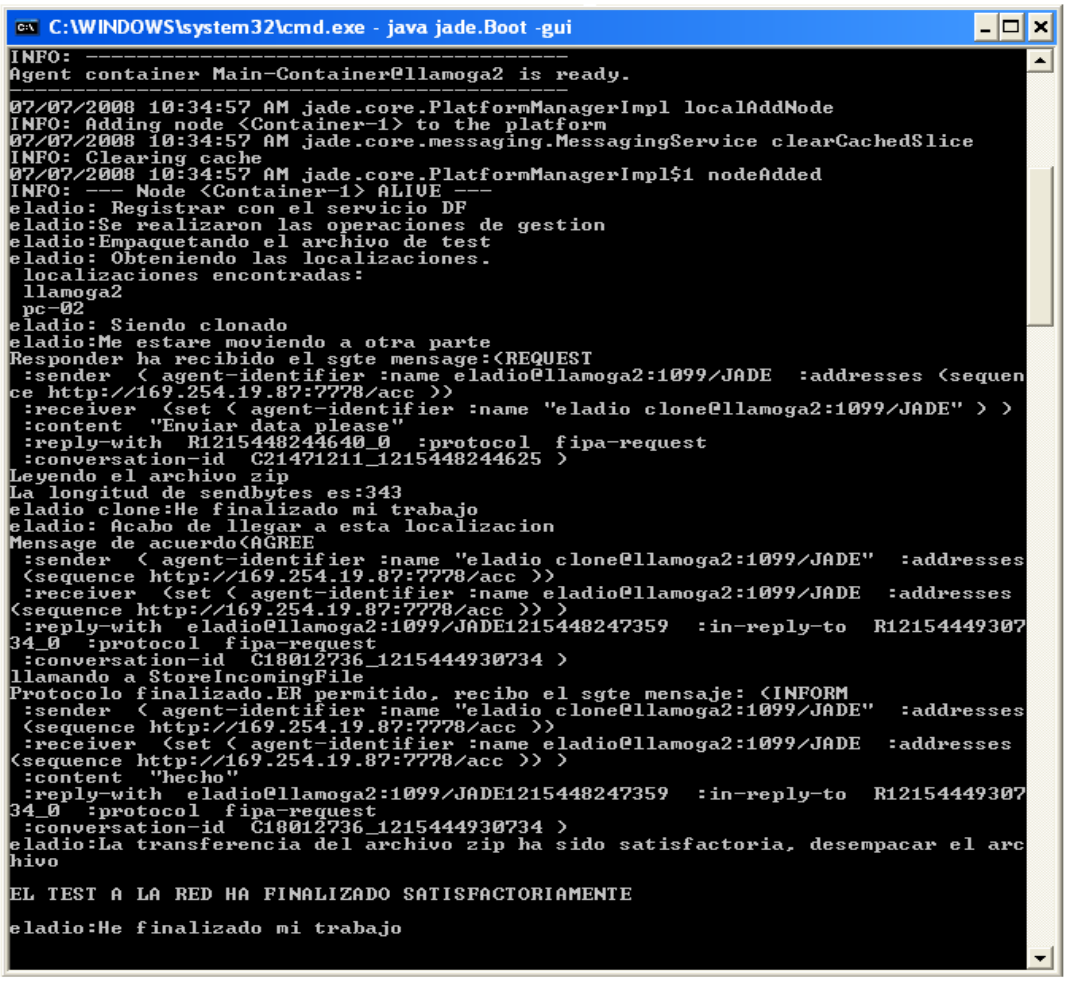


Figura 7.12 Lenguaje de comunicación ACL en el contenedor principal [ELLS].

- El contenido del mensaje: “Enviar data please” y “hecho”, mensajes enviados tanto por el agente original como por el agente clon, dentro del proceso de transferencia de información.
- El protocolo de Interacción: Representado por el protocolo fipa-request.
- El informe ó mensaje final “EL TEST HA SIDO REALIZADO SATISFACTORIAMENTE”, por el agente original, una vez terminada todo del proceso de gestión de manera satisfactoria.

b. En los Contenedores Secundarios

La información dentro de uno de los contenedores secundarios, es similar al del contenedor principal, como: la performativa, los mensajes, y el protocolo de interacción, ya que lo único que cambia es la posición del agente, debido a que éste se está desplazando por cada una de las localizaciones dentro de la red (Figura 7.13).

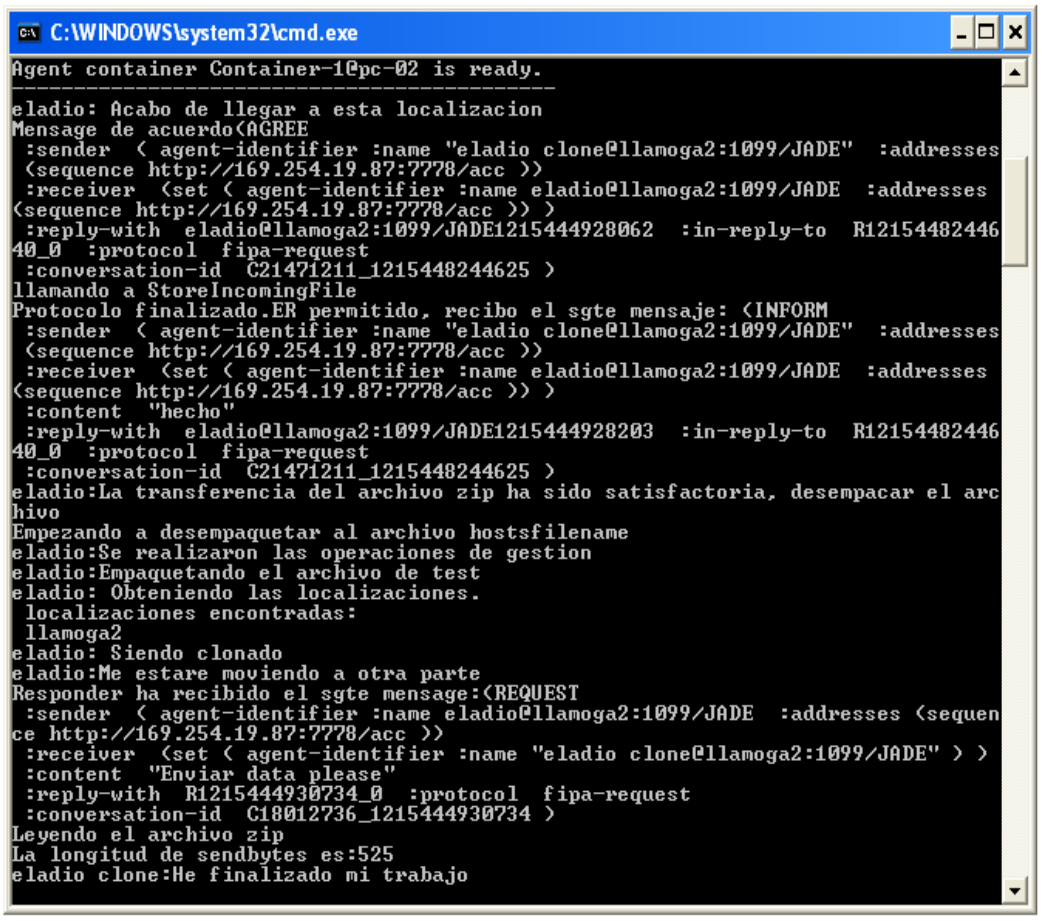


Figura 7.13 Lenguaje de comunicación ACL en el contenedor secundario [ELLS].

CAPITULO 8 Evaluación Comparativa de la Aplicación de Gestión de los Modelos (Centralizado vs Distribuido)

En este capítulo explicaremos un análisis comparativo de la aplicación de gestión aplicado dentro de un entorno centralizado, representado por “SNMPAgentTest”, y en el entorno distribuido, representado por el agente “AgenteMobilSNMP”.

8.1 EVALUACIÓN DEL TRÁFICO DE GESTIÓN DE AMBOS MODELOS

Tomando la siguiente consideración:

$$Tcs \geq T_{MA} \qquad (1)$$

De la ecuación (4) del capítulo 5.1 y la ecuación (6) del capítulo 7.5 tenemos la relación:

$$\sum_{n=1}^{N}\sum_{q=1}^{Q}(\eta_{cs}I_q + \acute{\eta}_{cs}R_{qn}) \geq 2\eta_{MA}C_{MA} + \acute{\eta}_{MA}(1-\sigma)R_1 + \acute{\eta}_{MA}\sum_{n=1}^{N+1}(1-\sigma)R_n \qquad (2)$$

Para la continuación del análisis y tener un mayor discernimiento, se considerará valores promedios donde:

Q: Número de instrucciones SNMP, para la obtención de objetos del MIB.

N: Número de estaciones gestionadas.

η_{cs}: Función de cabecera de la solicitud del mensaje, en el modelo C/S.

$\acute{\eta}_{cs}$: Función de cabecera de la réplica del mensaje, en el modelo C/S.

Iq: Ancho de la instrucción q^{th}. (Iq = I).

Rqn: Ancho de la replica q^{th} que viene del nodo n^{th}. (Rqn = R)

Rn: Resultado acarreados del nodo n^{th} .(Rn= P)

C_{MA}: Ancho del código móvil, incluyendo las instrucciones SNMP

η_{MA}: Función de cabecera del MA

$\acute{\eta}_{MA}$: Función de cabecera de las réplicas (resultados obtenidos)

La fórmula será re-escrita como:

$$NQ\eta_{cs}I + NQ\acute{\eta}_{cs}R \geq 2\eta_{MA}C_{MA} + \acute{\eta}_{MA}(1-\sigma)R1 + (N+1)\acute{\eta}_{MA}(1-\sigma)P \quad (3)$$

Estableciendo valores calculados en el apéndice D, los cuales van a permanecer en todo proceso de gestión son: I = 18 octetos, R = 23 octetos, P = 130 octetos (ancho del archivo recogido) y a una tasa de compresión de datos ($\sigma = 0.6$).
La notación anterior quedará establecida.

$$18NQ\eta_{cs} + 23NQ\acute{\eta}_{cs} \geq 2\eta_{MA}C_{MA} + 52\acute{\eta}_{MA} + 52(N+1)\acute{\eta}_{MA} \quad (4)$$

De las expresiones finales se deduce que:

1. A mayor cantidad de *estaciones* gestionados, mayor tráfico ocasionado en el modelo centralizado.
2. A mayor cantidad de *instrucciones SNMP*, mayor tráfico ocasionado en el modelo centralizado.

La siguientes tabla 8.1, cuyos datos han sido obtenidos de los capítulos 5.4 (entorno centralizado) y del capítulo 7.7.3 (entorno distribuido), confirma la relación de la ecuación (1).

UTILIZACION DE LA INTERFAZ		
ESTACIONES GESTIONADAS	ENTORNO CENTRALIZADO (%)	ENTORNO DISTRIBUIDO (%)
1	1.34403E-02	1.30995E-02
2	1.15752E-02	1.15649-02
3	1.87203E-02	1.85607E-02

Tabla 8.1 Tabla comparativa de la utilización de la interfaz

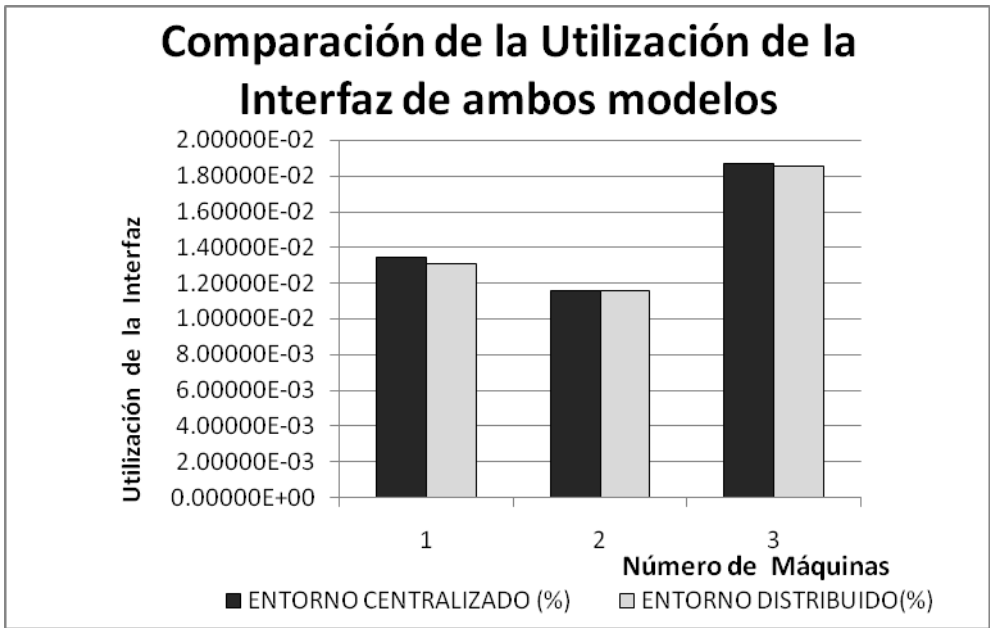


Figura 8.1 Gráfica comparativa de la utilización de la interfaz [ELLS].

- En la tabla 8.1, se puede observar que el porcentaje de Utilización de la Interfaz en el entorno centralizado es mayor que en el entorno distribuido, el cual nos indica que en dicho entorno, hay más cantidad de tráfico en la cola de la interfaz.
- A medida que aumenta el número de nodos gestionados, se aprecia mejor la diferencia de tráfico entre el modelo centralizado y distribuido.

8.2 EVALUACIÓN DEL TIEMPO DE GESTIÓN DE AMBOS MODELOS

Tomando la siguiente consideración:

$$Kt_{MA\ total} \geq Kt_{cs} \qquad (5)$$

De la ecuaciones (4) del capítulo 7.6 y la ecuación (5) del capítulo 5.2 tenemos:

$$\sum_{n=1}^{N}\lambda_{MAn} + \sum_{n=1}^{N}\frac{\eta_{MA}C_{MA}}{\tau_{MAn}} + \sum_{n=1}^{N}\lambda_{n} + \sum_{n=1}^{N}\frac{\acute{\eta}_{MA}S_{MA,n}}{\tau_{n}} + \sum_{n=1}^{N}\frac{\acute{\eta}_{MA}(1-\sigma)R_{n}}{\tau_{n}} \geq 14\lambda_{cs} + \sum_{q=1}^{Q}\frac{\eta_{cs}I_{q} + \acute{\eta}_{cs}R_{q}}{\tau_{cs}} \qquad (6)$$

Para la continuación del análisis y tener un mayor discernimiento, se considerará valores promedios, donde:

N: Número de nodos gestionados.

Q: Número de instrucciones SNMP.

λ_{MAn} : Retardo de transmisión del agente móvil, en el enlace n. ($\lambda_{MAn} = \lambda_{MA}$)

λ_n : Retardo de transmisión de resultado en el enlace n. ($\lambda_n = \lambda_R$)

τ_{MAn}: Throughput al arribar el agente móvil al nodo n. ($\tau_{MAn} = \tau_{MA}$)

τ_n: Throughput al arribar el resultado al nodo n. ($\tau_n = \tau_R$)

Además de: Iq = I, Rq = R, y Rn = P

Tendremos:

$$N\lambda_{MA} + \frac{N\eta_{MA}C_{MA}}{\tau_{MA}} + N\lambda_R + \sum_{n=1}^{N} \frac{\acute{\eta}_{MA}(n-1)(1-\sigma)P}{\tau_R} + \frac{N\acute{\eta}_{MA}(1-\sigma)P}{\tau_R} \geq 14\lambda_{CS} + \frac{Q\eta_{CS}I}{\tau_{CS}} + \frac{Q\acute{\eta}_{CS}R}{\tau_{CS}} \qquad (7)$$

Estableciendo valores calculados en el apéndice D, donde:

I = 18 octetos R = 23 octetos σ = 0.6

P = 130 bytes (ancho del archivo recogido).

La notación anterior quedará establecida como:

$$N\lambda_{MA} + \frac{N\eta_{MA}C_{MA}}{\tau_{MA}} + N\lambda_R + 52\sum_{n=1}^{N} \frac{\acute{\eta}_{MA}(n-1)}{\tau_R} + \frac{52N\acute{\eta}_{MA}}{\tau_R} \geq 14\lambda_{CS} + \frac{18Q\eta_{CS}}{\tau_{CS}} + \frac{23Q\acute{\eta}_{CS}}{\tau_{CS}} \qquad (8)$$

$$N(\lambda_{MA} + \lambda_R) + \frac{N\eta_{MA}C_{MA}}{\tau_{MA}} + 52\sum_{n=1}^{N} \frac{\acute{\eta}_{MA}(n-1)}{\tau_R} + \frac{52N\acute{\eta}_{MA}}{\tau_R} \geq 14\lambda_{CS} + \frac{18Q\eta_{CS}}{\tau_{CS}} + \frac{23Q\acute{\eta}_{CS}}{\tau_{CS}} \qquad (9)$$

De la expresión final se deduce que:

1. A mayor cantidad de nodos gestionados, mayor será el recorrido del agente móvil AMSNMP, y por lo tanto generará más tiempo de gestión.
2. A mayor cantidad de nodos gestionados, mayor será la información acarreada a través de los enlaces.
3. En el modelo distribuido, se observan dos tipos de retardos (el originado por el agente móvil, y el originado por los resultados obtenidos).
4. En el modelo distribuido, se observa también dos tipos de throughput (del MA, y el de los datos acarreados).

De esta evaluación, para un proceso de gestión, el tiempo empleado está en función de la cantidad de nodos gestionados, y a la cantidad de datos acarreados.

La siguientes tabla 8.2, cuyos datos han sido obtenidos de los capítulos 5.4 (entorno centralizado) y del capítulo 7.7.3 (entorno distribuido), confirman la relación de la ecuación (2).

TIEMPOS DE GESTION		
ESTACIONES GESTIONADAS	ENTORNO CENTRALIZADO (seg)	ENTORNO DISTRIBUIDO (seg)
NMS-1PC	5	3
NMS-2PC's	4	6
NMS-3PC's	5	12

Tabla 8.2 Tabla comparativa de los tiempos de gestión

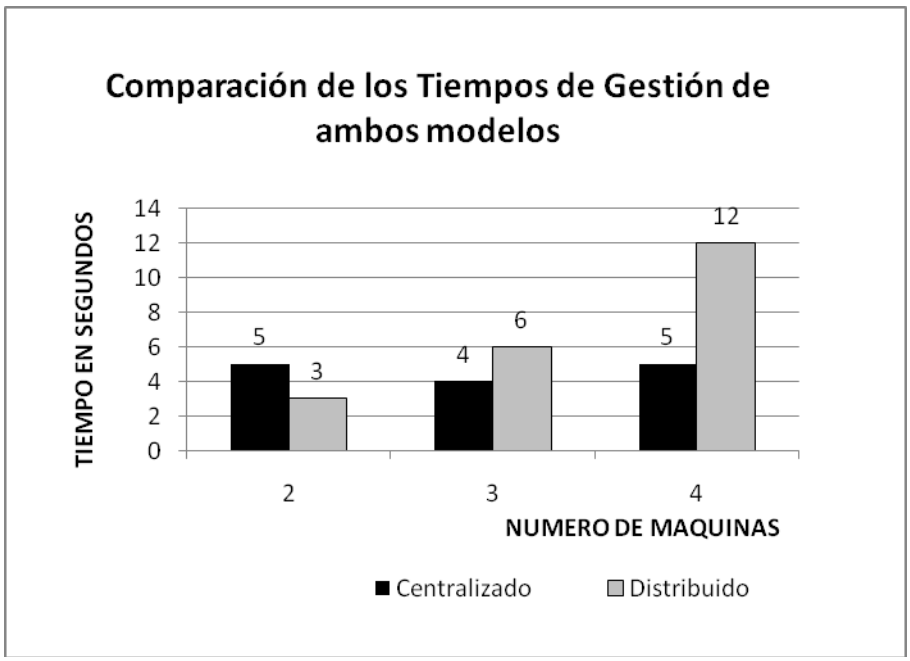


Figura 8.2 Gráfica comparativa de los tiempos de gestión [ELLS].

- En la tabla 8.2 se puede observar que los tiempos de gestión en el entorno centralizado permanece constante al incrementar el número de nodos gestionados, en relación a su contraparte, el entorno distribuido, donde el tiempo se va incrementando con la cantidad de nodos gestionados.

Del análisis comparativo y de los datos obtenidos, se puede tener una idea más inteligible de ambos modelos de gestión.

CAPITULO 9 Conclusiones y Trabajos Futuros

9.1 CONCLUSIONES

- La tecnología de agentes móviles, cuya estructura implementada en sistemas distribuidos, es un campo de investigación reciente, cuyo interés basado en el diseño orientado a objetos ofrecen de una manera natural e intuitiva de ver el proceso de gestión.
- Este trabajo de investigación aporta y propone una nueva forma de realizar el proceso de gestión mediante la aplicación de un agente móvil denominado “AgenteMobilSNMP –AMSNMP” y seguir los mismos elementos de operación y funciones distribuidos de gestión en el ámbito SNMP.
- Ninguna opción de los modelos de arquitecturas actuales ofrece una solución perfecta en términos de tráfico, sin antes tener un costo en el performance en la red, lo que nos lleva a decir que es imposible obtener datos sin afectar de una u otra forma la calidad de funcionamiento de toda red de telecomunicaciones.
- De los resultados cuantitativos obtenidos a través de pruebas realizadas durante el desarrollo de esta investigación, obtenemos que el esquema de

gestión centralizada ocasiona más tráfico en comparación con su contraparte, el agente móvil, esto es debido a que las decisiones de polling esta centralizado en la estación de gestión sobre los dispositivos a gestionar, en cambio en los agentes móviles los resultados son obtenidos a través de notificaciones de eventos realizados en cada dispositivo gestionado, y solo los resultados retornarán a la estación de gestión.

- El tiempo de gestión en el modelo de agente móvil se va incrementando a medida que aumentan las estaciones gestionadas, en comparación con su contraparte del modelo centralizado.

9.2 TRABAJOS FUTUROS

Dentro de los trabajos futuros, que pueden ser desarrollados a partir de esta investigación podemos mencionar:

- El problema de seguridad a la hora de gestionar, aunque JADE presenta un sistema de seguridad a través de identificaciones numéricas de manera aleatoria, y no a través de llaves públicas y privadas, así como de encriptación de la información.
- Implementar un sistema que permita que cada plataforma como los contenedores secundarios sean conectados directamente desde el contenedor principal, y no tener que ir a cada uno de los contenedores secundarios para realizar la operación de conexión a la plataforma.
- Implementar una capa de Inteligencia Artificial, en conjunción con otros lenguajes de programación, el cual permita a agente móvil, no solo muéstranos el performance de la red, sino de dar solución de manera rápida y eficiente, sin la intervención humana, en caso de que ocurra un problema en la red.
- Crear un sistema de gestión mediante la aplicación de agente móvil, el cual permita subdividir la red en varias subredes, y en el cual cada subred sea controlada por un agente, existiendo una comunicación entre ellos, y así reducir el tiempo de gestión, en redes alta envergadura.

BIBLIOGRAFÍA

[AdventNet04] AdventNet Inc, “Getting Started”, AdventNet SNMP 4, 2004.

[AdventNetSNMPv1] AdventNet Inc, “SNMP V1”, AdventNet SNMP 4, 2004.

[Aglets] Aglets. http://www.aglets.org/.

[BaldiPicco98] Baldi Mario, Picco Gian Pietro, “Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Aplications”, Departamento de Automatica e Informática de Torino, Italia, 1998.

[Bellifemine03] Bellifemine Fabio, et al., “JADE - A White Paper”, Exp. Búsqueda de la innovación, vol. 3-n.3, Setiembre 2003.

[Bellifemine07] Bellifemine Fabio, et al., “Development multi-agent systems with JADE” Jhon Willey & Sons Ltd, Inglaterra, 2007.

[Bellifemine08] Bellifemine Fabio, et al., “JADE Programmer’s Guide”, TILAB CSELT, Junio 2007.

[Bierman98] Bierman Andy, et al., “Distributed Management Framework”, Cisco Systems, Agosto 1998.

[Boutaba] Boutaba Raouf, Xiao Jin, “Network Management : State of the Art”, Department of Computer Science, University of Waterloo, Canadá.

[Caire08] Caire Giovanni, “JADE TUTORIAL: JADE Programming for Beginners”, TILAB CSELT, Setiembre 2007.

[Calvo01] Calvo Isidro, “Seminario de CORBA”, Departamento de Ingeniería de Sistemas y Automática, Mayo 2001.

[CORBA] OMG Document Number: formal/2008-01-04, "Common Object Request Broker Architecture", Version 3.1, Enero 2008. http://www.omg.org/spec/CORBA/3.1/Interfaces/pdf.

[Cisco03] Cisco - Performance Management: Best Practices White Paper, 2003.

[Deitel04] Deitel Harvey M, Deitel Paul J, "Como programar en Java". Pearson Educación, México 2004, Quinta Edición.

[dit00] Departamento de Ingeniería de Sistemas Telemáticos, "Aspectos Funcionales de Gestión de Red", Universidad Politécnica de Madrid 2000.

[FIPA] Foundation for Intelligent Physical Agents. http://www.fipa.org/.

[FIPA008] Foundation for Intelligent Physical Agents, "FIPA SL Content Language Specification", Marzo 2002.

[FIPA023] Foundation for Intelligent Physical Agents, "Agent Management Specification", Marzo 2004.

[FIPA026] Foundation for Intelligent Physical Agents, "FIPA Request Interaction Protocol Specification", Marzo 2002.

[FIPA061] Foundation for Intelligent Physical Agents, "FIPA ACL Message Structure Specification", Marzo 2002.

[FIPA087] Foundation for Intelligent Physical Agents, "FIPA Agent Management Support for Mobility Specification", Octubre 2001.

[Fountoukidis04] Fountoukidis Dimitrios P, "Adaptive Management of Emerging Battlefield Network", Naval Postgraduate School, Monterrey, California, Marzo 2004.

[GDMO] Agudelo Oscar, "Guidelines for Definition of Managed Objects", 2001.

[GestionOSI] "Modelos de Gestión de red". Tema3.

[Gómez99] Gómez Labrador, Ramón M., “Agentes Móviles y CORBA”, Universidad de Sevilla, Febrero 1999.

[Grasshopper] Grasshopper Agent Platform, http://www.grasshopper.de/

[Hayes-Roth] Hayes-Roth Barbara, “An Architecture for Adaptive Intelligent System”, Knowledge Systems Laboratory, Stanford University, Palo Alto, California.

[Hegering] Heinz-Gerd Hegering, et al., “Integrated Management of Network System”, Morgan Kaufmann 1995.

[Juarez05] Juárez Valdez Marcel Isabel, Seguridad en Sistemas Multi-agentes, Tesis, Ensenada, Baja California México, p.20 Diciembre 2005.

[LangeOshima98] Lange Danny B, Oshima Mitsuru, “Mobile Agents with Java: The Aglet API”, Generic Electric Inc, IBM Tokyo Research, 1998.

[Liotta99] Liotta A, Knight G, Pavlou G, “On the Efficiency Scalability of Decentralized Monitoring using Mobile Agents”, University London, University of Surrey, 1999.

[LopesOliveira00] Lopes Rui P, Oliveira José Luis, “On the use of Mobility in Distributed Network Management”, 33rd *Hawaii International Conference on System Sciences*, 2000.

[Maes] Maes Pattie “Modeling Adaptive Autonomous Agents”, MIT Media Laboratory.

[Makki06] Makki Shamila, Wunnava Subbarao, “Application of Mobile Agents in Managing the Traffic in the Network and Improving the Reliability and Quality of Service”, IAENG *International Journal of Computer Science*, Noviembre 2006.

[Mota] Mota Telma, et al., “Quality of Service Management in IP Network using Mobile Agent Tecnology”.

[Nguyen02] Nguyen G, et al., “Agent Platform evaluation and comparison”, Institute of Informatics, Slovak Academy of Sciences, Junio 2002.

[OMG] Object Management Group, http:// www.omg.org/.

[O’Reilly01] Douglas Mauro, Kevin Schmidt. “Essential SNMP” O’Reilly, Julio 2001.

[Pras99] Pras Aiko, et al., “Introduction to TMN”, CITT Technical Report, University of Twenty, Abril 1999.

[Pras04] Pras Aiko et al., “Comparing the Performance of SNMP and Web Services-Based Management, 2004.

[Remick02] Brian D. Remick “Managing Agent Platform with Simple Network Protocol”, Universidad de Utah.

[RFC1155] Rose M, McCloghrie K, “Structure and Identification of Management Information for TCP/IP – based Internets, Network Working Group, Mayo, 1990.

[RFC1156] McCloghrie K, Rose M, “Management Information Base for Network Management of TCP/IP – based Internet, Network Working Group, Mayo 1990.

[RFC1157] Case J, et al., “A Simple Network Management Protocol (SNMP)”, Network Working Group, Mayo 1990.

[RFC1213] McCloghrie K, Rose M, “Management Information Base for Network Management of TCP/IP – based Internet: MIB II, Network Working Group, Marzo 1991.

[RFC 2386] RFC 2386 “A framework for QoS-based Routting in the Internet”, Agosto 1998.

[RFC2573] Levi D., Meyer P, “SNMP Applications”, Network Working Group, April 1999.

[RFC2576] Frye R., Levi D., “Coexistence between version 1, version 2, and version 3 of the Internet Standard Network Management Framework”, Marzo 2000.

[RFC3231] Levi D, Schoenwaelder J, “Definitions of Managed Objects for Scheduling Management Operations”, Enero 2002.

[RFC3780] Strauss F., Schoenwaelder J., “SMIng – Next Generation Structure of Management Information”, Grupo de Trabajo de Red, Mayo 2004.

[Riu02] Rui P. Sanches de Castro Lopes “ Gestão Distribuida em SNMP”, Universidad de Aveiro, 2002.

[Schildt05] Helbert Schildt, “ La Biblia de Java 2 v5.0” Grupo ANAYA, S.A. 2005.

[Schoder00] Schoeder Detlef, Eymann Torsten, “The Real Challenges of Mobile Agents”, Communications of the ACM, vol43, N°6, Junio, 2000.

[Shoham] Shoham Yoav, “Agent-oriented Programming”, 1993.

[SNMPWorks] Windows 2003, Marzo 2003. http://technet.microsoft.com/en-us/library/

[Straber] Straber Markus, Schwehm Markus, “A Performance Model for Mobile Agent System, University of Stuttgart, Alemania.

[Steffen02] Steffen Andreas, “Abstract Syntax Notation One – ASN.1”, Zürcher Hochschule Winterthur, 2002.

[Sun_Min06] Sun-Mi Yoo, et al., “Performance Evaluation of WBEM Implementation” Internet Server Group, Daejeou, Korea, Febrero 2006.

[Tarr00] Tarr Bob, et al., “Introduction to Mobile Agent Systems and Applications”, Department Defense-USA, 2000.

[Vaucher03] Vaucher J, Ncho A, “JADE Tutorial and Primer”, 2003.

[Vergara03] Lopez de Vergara Méndez José Enrique, “Especificación de Modelo de Información de Gestión de Red Integrada mediante el uso de Ontologías y Técnicas de representación de conocimiento”, Universidad Politécnica de Madrid, 2003.

[Wooldridge] Wooldridge Michael, Nicholas R. Jennings, "Intelligent Agents: Theory and Practice” Manchester Metropolitan University, UK, Queen Mary & Westfield College, UK, Enero 1995.

[X.700] Recommendation X.700, “Management Framework for Open Systems Interconnection (OSI) for CCITT Application, Setiembre 1992.

[X.701] Recommendation X.701, “Information Tecnology – Open Systems Interconnection - Systems Management Overview, Enero 1992.

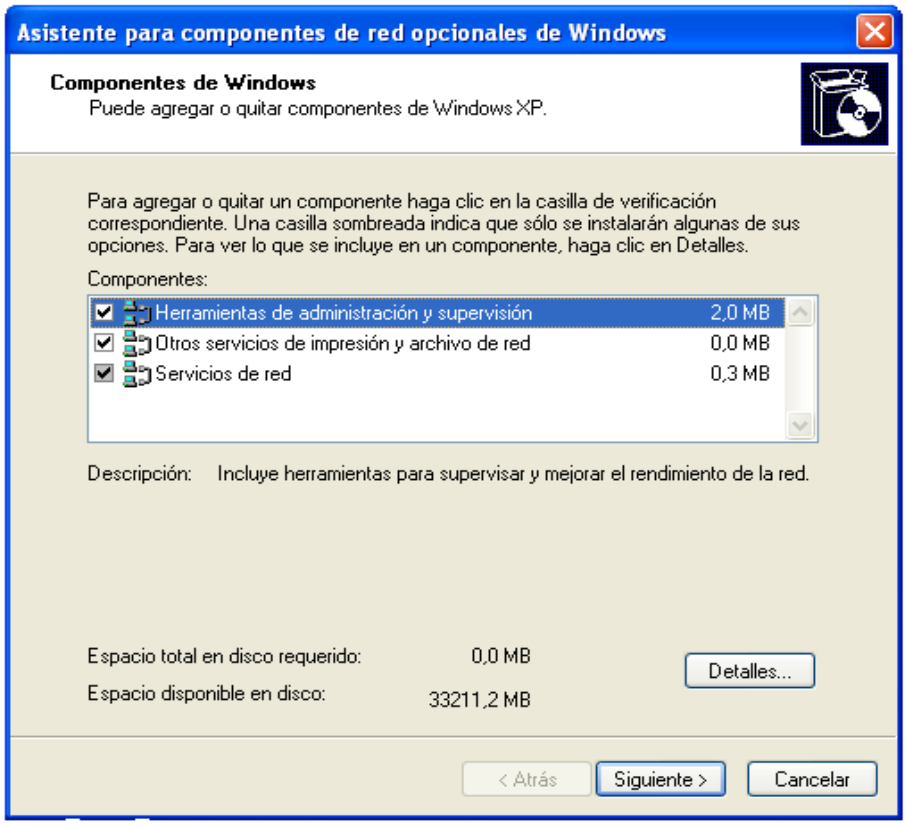
Anexo A.

INSTALACIÓN Y CONFIGURACIÓN DEL AGENTE SNMP

1. Para la instalación:

Sea el caso que no esté instalado dicho agente SNMP.

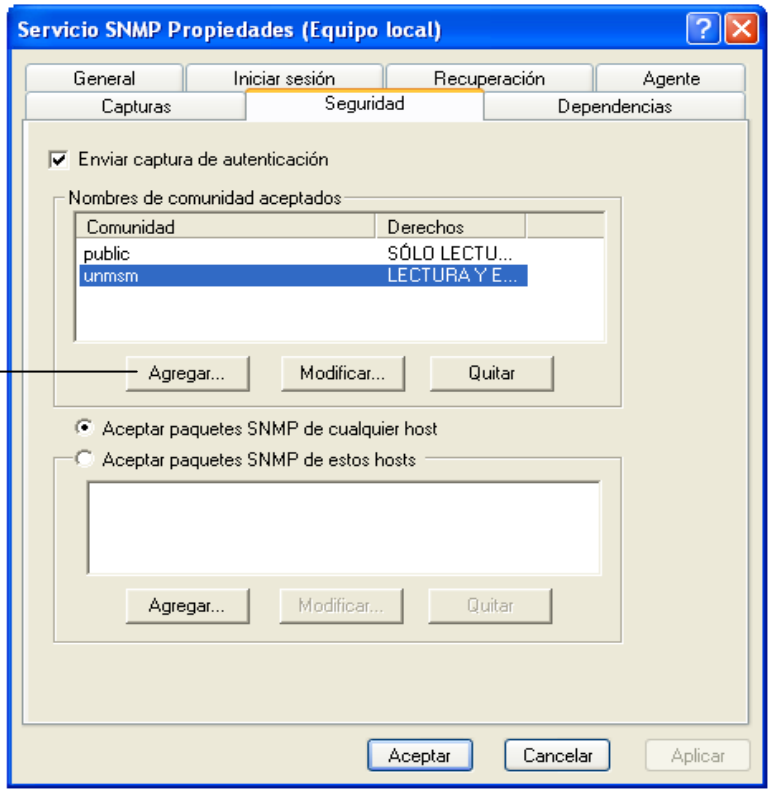
Seleccionar ‘Propiedades’ en mi sitio de red, luego ir a ‘Opciones Avanzadas’ y seleccionar “Componentes de red opcionales” (Figura 1) y de allí, habilite la opción ‘Herramientas de Administración y Supervisión’. Después de esto el Sistema Operativo solicitará la introducción de CD de Windows XP.



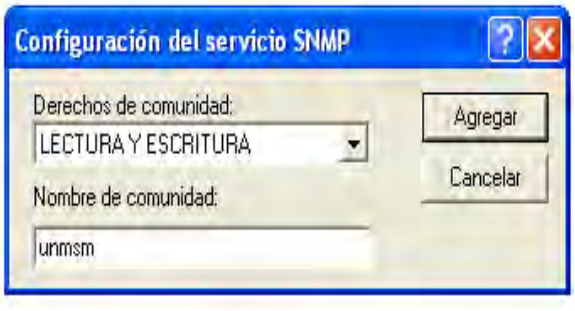
Anexo A: Figura1. Instalación de las herramientas de gestión SNMP [ELLS].

2. Habilitar el Servicio SNMP

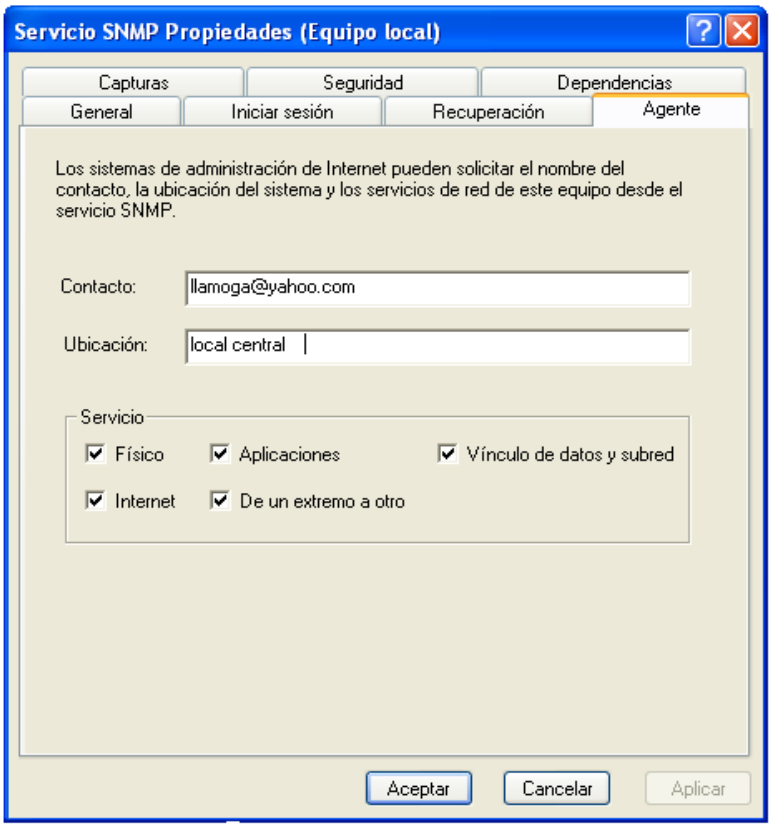
Seleccionar “Panel de Control” dentro de Configuración, luego seleccionar Herramientas Administrativas, para ingresar a “Servicios”, y dentro de ella seleccionar ‘Servicio SNMP’ y establecer los parámetros que han sido establecidos en la aplicación como: la comunidad al cual denominamos ‘unmsm’ (Figura 1), así como el contacto y ubicación del sistema (Figura 2)



Anexo A: Figura 2. Establecimiento de la comunidad [ELLS].



Y por último configurar del agente, en el cual se especifica el contacto y la ubicación del sistema.



Anexo A: Figura 3. Configuración del agente SNMP [ELLS].

Anexo B.

CÓDIGO JAVA DE LA APLICACIÓN DE GESTIÓN EN EL ENTORNO CENTRALIZADO

```
/** Haciendo un test al agente SNMP, para calcular la utilización de la Interface,
 y la tasa de error,  en un entorno de gestión centralizada.
* Autor: Eladio Llamoga Sánchez.
* Universidad Nacional Mayor de San Marcos.
* Version.1.0 */
import java.net.*;
import java.util.*;
import java.io.*;
import com.adventnet.snmp.beans.*;
import com.adventnet.snmp.mibs.*;
public class SNMPAgentTest {
String host = null;
String comunidad;
double valor1, valor2, valor3, valor 4, valor 5, valor6, valor7;
public SNMPAgentTest(String hostAddress){
host = hostAddress;}
public void valueTest(boolean set){
try{
System.out.println ("Comienzo del Test");
SnmpTarget target = new SnmpTarget();
target.setTargetHost(host);
target.setWriteCommunity("unmsm");
target.loadMibs("\"C:/Archivos de programa/AdventNet/mibs/RFC1213-MIB\"");
if(set)target.snmpSet("Esto es un test");
target.setObjectID(".1.3.6.1.2.1.2.2.1.10.2"); //ifInOctects
String resultado1 = target.snmpGet();
if(resultado1!=null){ valor1= Double.parseDouble(resultado1);}
else{System.out.println("Error del valor1");}
target.setObjectID(".1.3.6.1.2.1.2.2.1.16.2"); //ifOutOctects
String resultado2 = target.snmpGet();
```

```
if(resultado2!=null){ valor2=Double.parseDouble(resultado2);}
else{System.out.println("Error del valor2");}
target.setObjectID(".1.3.6.1.2.1.2.2.1.5.2");  //ifSpeed
String resultado3 = target.snmpGet();
if(resultado3!=null){valor3=Double.parseDouble(resultado3);}
else{System.out.println("Error en el valor3");}
target.setObjectID(".1.3.6.1.2.1.2.2.1.14.2");        //ifInError
String resultado5 = target.snmpGet();
if(resultado5!=null){valor5=Double.parseDouble(resultado5);}
else{System.out.println("Error en el valor5");}
target.setObjectID(".1.3.6.1.2.1.2.2.1.11.2");        //ifInUcastPkts
String resultado6 = target.snmpGet();
if(resultado6!=null){valor6=Double.parseDouble(resultado6);}
else{System.out.println("Error en el valor6");}
target.setObjectID(".1.3.6.1.2.1.2.2.1.12.2");        //ifInNUCastPkts
String resultado7 = target.snmpGet();
if(resultado7!=null){valor7=Double.parseDouble(resultado7);}
else{System.out.println("Error en el valor7");}
// Conversión de días, horas, minutos a segundos, en centésimas de segundos
SnmpOID oid = new SnmpOID("1.3.0");
SnmpVar snmpvar = target.snmpGet(oid);
SnmpTimeticks s = (SnmpTimeticks)snmpvar;
double valor4 = (double)s.longValue();
// Los valores de los objetos MIB son:
System.out.println("El host Address es:" +target.getTargetHost());
 System.out.println("El ifInOctets es:"+valor1+", "+"El ifOutOctets es:"+valor2+", "+"El
 ifSpeed es:"+valor3+", "+"El ifInError es:"+valor5+", "+"El ifInUcastPkts es:"+valor6+",
 "+"El ifInNUCastPkts es:"+valor7+", "+"El sysUpTime es:"+valor4);
System.out.println("Fin del Test");
Date date = new Date();
SimpleDateFormat sdf;
sdf = new SimpleDateFormat ("dd:mm:ss");
System.out.println(sdf.format(date));
target.releaseResources();}
catch(Exception e) {System.out.println(e.toString());}
```

```
}
//Limpieza de registro
public void resetSysName(){
DataInputStream is=null;
try{    Runtime r =Runtime.getRuntime();
        try{Process helpProcess = r.exec("reg delete
        hklm\\System\\CurrentControlSet\\Services\\SNMP\\Parameters\\RFC1156Agent /v
        SysName/f");
if(helpProcess == null){
System.out.println("No se puede resetear el registro");}
is= new DataInputStream(helpProcess.getInputStream());
        }
catch(IOException ioe){
System.err.println("IOException:" +ioe);}
BufferedReader br = new BufferedReader(new InputStreamReader(is));
String responseLine;
while((responseLine = br.readLine())!=null){
System.out.println("Server:" + responseLine);}
is.close();}
catch(IOException e){
System.err.println("IOException:" + e);
        }
}
public static void main (String args[]){
String hostAddress[] = {"llamoga2","pc-02","pc-06", “pc-08”};
SNMPAgentTest s0 = new SNMPAgentTest(hostAddress[0]);
SNMPAgentTest s1 = new SNMPAgentTest(hostAddress[1]);
SNMPAgentTest s2 = new SNMPAgentTest(hostAddress[2]);
SNMPAgentTest s3 = new SNMPAgentTest(hostAddress[3]);
SNMPAgentTest saget[] = {s0, s1, s2,s3};
for(int n=0; n<saget.length; n++){
saget[n].valueTest(false);}
System.exit(0);
        }
}
```

Anexo C.

CÓDIGO JAVA DEL AGENTE MÓVIL SNMP - AMSNMP EN EL ENTORNO DISTRIBUIDO

```
/** Agente móvil SNMP  para calcular la utilización de la interfaz, y la tasa de error,  dentro
        de un entorno de gestión distribuido.
* Autor: Eladio Llamoga Sánchez.
* Universidad Nacional Mayor de San Marcos.
* Version.1.0 */
package tesis;
import jade.core.*;
import jade.core.behaviours.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import jade.content.*;
import jade.content.onto.basic.*;
import jade.content.lang.Codec;
import jade.content.lang.sl.SLCodec;
import jade.domain.mobility.MobilityOntology;
import jade.domain.JADEAgentManagement.*;
import jade.lang.acl.*;
import jade.proto.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.zip.*;
import tesis.util.*;
import com.adventnet.snmp.beans.*;
import com.adventnet.snmp.snmp2.*;
import com.adventnet.snmp.mibs.*;

public class AgenteMobilSNMP extends Agent {
```

```
private String agenteSNMP = "file.txt";
private String agentetipo = "Tipo Mobil";
private String hostsfilename = "llamoga.txt";
private String senddata = "Enviar data please";
private String zipfilename = "archivo.zip";
private String dirfiles = "files";
private boolean cloned = false;
private boolean agenteoriginal = false;
private boolean returnhome = false;
protected boolean error = false; // si hay cualquier error
protected boolean checkpoint = false;
protected String checkhostname;
protected String hosthome;
protected byte[] data;
private transient Location locationhost;
protected void setup(){
agenteoriginal=true;
//Obteniendo la dirección local de tipo Location
hosthome = here().getAddress();
//Registrar el Lenguaje y la Ontologia para este host
getContentManager().registerLanguage(new  SLCodec());
getContentManager().registerLanguage(newSLCodec());,
        FIPANames.ContentLanguage.FIPA_SL0);
getContentManager().registerOntology(MobilityOntology.getInstance());

try {
System.out.println(getLocalName()+": Registrar con el servicio DF");
DFAgentDescription dfad = new DFAgentDescription();
dfad.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType(agentetipo);
sd.setName(getName());
dfad.addServices(sd);
DFService.register(this, dfad);
                } catch (Exception e) {
```

```
System.err.println(getLocalName()+": Error al registrar con el servicio DF");
error=true;
doDelete();
                    }
addBehaviour( new ChequearHostPrincipal(this) );
}
//Este behaviour realiza la búsqueda del host principal en la red,
//realiza el test y el resultado será almacenado en un archivo de texto y luego
//empaquetado para su traslado
class ChequearHostPrincipal extends OneShotBehaviour{
public ChequearHostPrincipal(Agent a){
super(a);}
public void action(){
try{
BufferedReader br1 = new BufferedReader(new FileReader(hostsfilename));
if( (checkhostname = br1.readLine()) != null){
br1.close();
                }
else{System.err.println(getLocalName() +":Error al buscar el archivo llamoga.txt");
error= true;
myAgent.doDelete();}
//chequea si estamos en el host principal
checkpoint = hosthome.equals(checkhostname);
if(checkpoint){
addBehaviour(new SNMPAgentTest(myAgent));
addBehaviour(new EmpaquetarFiles(myAgent, true));
}
else{addBehaviour(new EmpaquetarFiles(myAgent, false));}
addBehaviour( new ObtenerLocalizaciones(myAgent) );
}
catch(Exception e){System.err.println(getLocalName()+": Error Buscando Hosts");
error=true;
myAgent.doDelete();}
                    }
}
```

```
//Behaviour que realiza la operación de gestión
class SNMPAgentTest extends OneShotBehaviour{
private String host = null;
private String comunidad;
private double valor1,valor2,valor3,valor4,valor5,valor6,valor7;
InetAddress direccion;
public SNMPAgentTest(Agent a){
super(a);
        }
public void action(){
try{
direccion = InetAddress.getLocalHost();
host = direccion.getHostName();
SnmpTarget target = new SnmpTarget();
target.setTargetHost(host);
target.setWriteCommunity("unmsm");
target.loadMibs("\"C:/Archivos de programa/AdventNet/mibs/RFC1213-MIB\"");
//if (true)target.snmpSet("Esto es un test");
target.setObjectID(".1.3.6.1.2.1.2.2.1.10.2");      //ifInOctects
String resultado1 = target.snmpGet();
if(resultado1!=null){valor1 = Double.parseDouble(resultado1);}
else{System.out.println("Error del valor1");}
target.setObjectID(".1.3.6.1.2.1.2.2.1.16.2");      //ifOutOctects
String resultado2 = target.snmpGet();
if(resultado2!=null){valor2 = Double.parseDouble(resultado2);}
else{System.out.println("Error del valor2");}
target.setObjectID(".1.3.6.1.2.1.2.2.1.5.2");       //ifSpeed
String resultado3 = target.snmpGet();
if(resultado3!=null){valor3 = Double.parseDouble(resultado3);}
else{System.out.println("Error del valor3");}
target.setObjectID(".1.3.6.1.2.1.2.2.1.14.2");      //ifInError
String resultado5 = target.snmpGet();
if(resultado5!=null){valor5 = Double.parseDouble(resultado5);}
else{System.out.println("Error del valor5");}
target.setObjectID(".1.3.6.1.2.1.2.2.1.12.2");      //ifInUCastPkts
```

```
String resultado7 = target.snmpGet();
if(resultado7!=null){valor7 = Double.parseDouble(resultado7);}
else{System.out.println("Error del valor7");}
//Conversión de días, horas, minutos a segundos,
SnmpOID oid = new SnmpOID("1.3.0");
SnmpVar snmpvar = target.snmpGet(oid);
SnmpTimeticks s = (SnmpTimeticks)snmpvar;
double valor4 = (double)s.longValue();
//Los valores de los objetos MIB son:
 String resultadoTotal = "El ifInOctets es:"+valor1+", "+"El ifOutOctets es:"+valor2+",
 "+"El ifSpeed es:"+valor3+", "+"El ifInError es:"+valor5+", "+"El ifInUcastPkts
 es:"+valor6+", "+"El ifInNUCastPkts es:"+valor7+", "+"El sysUpTime es:"+valor4;
if(resultadoTotal!=null){System.out.println(getLocalName()+":Se realizaron las operaciones
        de gestión");
}
else{System.out.println(getLocalName()+":No se realizaron las operaciones de gestión");}
data = resultadoTotal.getBytes();
OutputStream fos = new FileOutputStream(agenteSNMP);
fos.write(data);
fos.close();
data = null;
java.lang.System.gc();
}catch(Exception e){e.printStackTrace();
                        error = true;
                        myAgent.doDelete();}
        }
}
//clase que empaqueta los resultados de la operación de gestión
class EmpaquetarFiles extends OneShotBehaviour{
boolean agregar;
public EmpaquetarFiles(Agent a, boolean agregar){
super(a);
this.agregar = agregar;}
public void action(){
try{
```

```
System.out.println(getLocalName()+":Empaquetando el archivo de test");
File f9 = new File(zipfilename);
boolean existe = f9.canRead();
EmpaquetarDesempaquetar ed;
if(existe){
ed = new EmpaquetarDesempaquetar(zipfilename, "add files");
                }
else{
ed = new EmpaquetarDesempaquetar(zipfilename, "new files");
ed.addPaquete(hostsfilename, hostsfilename);
                }
if(agregar){
ed.addPaquete(agenteSNMP, myAgent.here().getAddress() +".txt");}
ed.finDelEmpaquetamiento();
}catch(Exception e){
System.err.println(getLocalName()+":Error de empaquetamiento");
error = true;
myAgent.doDelete();
                        }
        }
}

//Este behaviour solicita los posibles destinos al agente AMS
class ObtenerLocalizaciones extends OneShotBehaviour{
public ObtenerLocalizaciones(Agent a){
super(a);
        }
public void action(){
try{boolean found=false;
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.addReceiver(getAMS());
msg.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
msg.setOntology(MobilityOntology.NAME);
msg.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
Action action = new Action();
```

```
action.setActor(getAMS());
action.setAction(new QueryPlatformLocationsAction());
getContentManager().fillContent(msg, action);
send(msg);
MessageTemplate mt = MessageTemplate.and(
MessageTemplate.MatchSender(getAMS()),
MessageTemplate.MatchPerformative(ACLMessage.INFORM));
mt=MessageTemplate.and(mt,
        MessageTemplate.MatchOntology(MobilityOntology.NAME));
mt=MessageTemplate.and(mt,
        MessageTemplate.MatchProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST))
        ;
mt=MessageTemplate.and(mt,
        MessageTemplate.MatchLanguage(FIPANames.ContentLanguage.FIPA_SL0) );
msg = blockingReceive(mt); //Si el ams no responde el agente se detendra
Result result = (Result)getContentManager().extractContent(msg);
jade.util.leap.Iterator it = result.getItems().iterator();
if(checkpoint){
BufferedReader br2 = new BufferedReader(new FileReader(hostsfilename));

//Verificando si hay otros hosts en el archivo hostsfilename
String readhostname=null;
do{
readhostname = br2.readLine();
}while(!checkhostname.equals(readhostname) && readhostname!=null);
checkhostname = br2.readLine();
br2.close();
}
checkpoint = true;
if(checkhostname == null){checkhostname = new String(hosthome);
returnhome=true;
}
//Si es el último host home entonces es el último, y hay que finalizar la operación
if (returnhome && hosthome.equals(myAgent.here().getAddress())){
myAgent.doDelete();
```

```
}
else{
//Adjuntar la localización con el próximo host dentro de la lista de archivo hostsfilename
locationhost = null;
System.out.println(getLocalName()+": Obteniendo las localizaciones.\n localizaciones
        encontradas:");
while (it.hasNext() && !found) {
locationhost = (Location)it.next();
System.out.println(" "+locationhost.getAddress());
found = locationhost.getAddress().equals(checkhostname);          }
if (!found){
System.err.println( getLocalName()+": Error al buscar el host "+checkhostname+".
        ABORTAR");
error=true;
myAgent.doDelete();}
else{
System.out.println(getLocalName()+": Siendo clonado");
//Se añade un behaviour que block(), hasta que la operación haya finalizado
addBehaviour(new Migracion(myAgent));
doClone(myAgent.here(), getLocalName()+" clone");}
        }
}catch(Exception e){e.printStackTrace();
error=true;
myAgent.doDelete();}
}
}

//Un behaviour que permitirá al agente original migrar al próximo host
class Migracion extends SimpleBehaviour{
public Migracion(Agent a){super(a);}
public void action(){
//Verifica si el agente ha sido clonado
if (cloned && agenteoriginal){myAgent.doMove(locationhost);}
else{block();}        //bloquea al behaviour esperando un nuevo evento
   }
```

```
//Finaliza el behaviour si el agente ha sido clonado
public boolean done(){
return (cloned);}
}

protected void beforeMove() {
System.out.println(getLocalName()+":Me estare moviendo a otra parte");}
protected void afterMove() {
cloned=false;
System.out.println(getLocalName()+": Acabo de llegar a esta localizacion");
//Registrando otra vez el lenguaje y la ontología de movilidad jade
//ya que estas variables no migran
getContentManager().registerLanguage(new SLCodec());
getContentManager().registerLanguage(newSLCodec(),
        FIPANames.ContentLanguage.FIPA_SL0);
getContentManager().registerOntology(MobilityOntology.getInstance());
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
request.addReceiver(new AID(getLocalName()+" clone", AID.ISLOCALNAME));
request.setContent(senddata);
addBehaviour(new TransferRequest(this, request));}
protected void beforeClone() {
try{    cloned=true;
}catch(Exception e){
System.err.println(getLocalName()+": Error de migracion. ABORTAR");
                }
}
protected void afterClone() {
try{    agenteoriginal=false;
        File hostfile = new File(hostsfilename);
        hostfile.delete();
MessageTemplate mt=
        AchieveREResponder.createMessageTemplate(FIPANames.InteractionProtocol.FIPA_
        REQUEST);
this.addBehaviour(new TransferReply(this, mt));
```

```
}catch(Exception e) {System.err.println(getLocalName()+":Error enviando el archivo");
error= true;
doDelete(); }

}
//Este behaviour almacena el archivo en el agente original, transferido por el clone
class TransferReply extends AchieveREResponder{
public TransferReply(Agent a, MessageTemplate mt){
super(a, mt);}
protected ACLMessage prepareResponse(ACLMessage request){
ACLMessage response = request.createReply();
response.setPerformative(ACLMessage.AGREE);
return response;}
protected ACLMessage prepareResultNotification(ACLMessage request, ACLMessage
      response){
System.out.println("Responder ha recibido el sgte mensage:" + request);
ACLMessage informDone = null;
try{
FileInputStream fis = new FileInputStream(zipfilename);
System.out.println("Leyendo el archivo zip");
ACLMessage filemessage = new ACLMessage(ACLMessage.UNKNOWN);
filemessage.addReceiver(request.getSender());
byte[] databytes = new byte[8192];
byte[] sendbytes;
int readbytes;
while((readbytes = fis.read(databytes))!=-1){
sendbytes = new byte[readbytes];
for(; readbytes!=0;readbytes--){
sendbytes[readbytes-1] = databytes[readbytes-1];}
System.out.println("La longitud de sendbytes es:"+sendbytes.length);
filemessage.setContentObject(sendbytes);
send(filemessage);
}
informDone = request.createReply();
informDone.setPerformative(ACLMessage.INFORM);
```

```
informDone.setContent("hecho");

File zipfile2 = new File(zipfilename);
zipfile2.delete();

// Como todo ha terminado, añadir un behaviour que delete al agente clon
addBehaviour(new OneShotBehaviour(myAgent){
public void action(){
myAgent.doDelete();}});
}catch(Exception e){e.printStackTrace();
                    error = true;
                    doDelete();}

return informDone;}
}
class TransferRequest extends AchieveREInitiator{
BufferedInputStream is1,is2,is3;
ZipEntry entry1,entry2,entry3;
ZipFile zipfile1,zipfile2,zipfile3;
FileOutputStream filefos;
StoreIncomingFile sifb;
public TransferRequest(Agent a, ACLMessage request){
super(a, request);}
protected void handleAgree(ACLMessage agree){
System.out.println("Mensage de acuerdo" + agree);
try{
filefos = new FileOutputStream(zipfilename);
System.out.println("llamando a StoreIncomingFile");
sifb = new StoreIncomingFile(myAgent, filefos);
myAgent.addBehaviour(sifb);
}catch(Exception e){e.printStackTrace();
                    error = true;
                    doDelete();}
}
```

```
protected void handleInform(ACLMessage inform){
System.out.println("Protocolo finalizado.ER permitido, recibo el sgte mensaje: "+inform );
try{
if(inform.getContent().equals("hecho")){
System.out.println(getLocalName() +":La transferencia del archivo zip ha sido satisfactoria,
        desempacar el archivo");
if(!returnhome){
System.out.println("Empezando a desempaquetar al archivo hostsfilename");

        ZipEntry entry1;
        ZipFile zipfile1 = new ZipFile(zipfilename);
        entry1 =zipfile1.getEntry(hostsfilename);
if(entry1!=null){
is1 = new BufferedInputStream(zipfile1.getInputStream(entry1));
int readbytes;
byte data[] = new byte[8192];
FileOutputStream fos1 = new FileOutputStream (entry1.getName());
while((readbytes = is1.read(data))!=-1){fos1.write(data,0,readbytes);}
is1.close();
fos1.close();
                }

        }
}
else{System.err.println(getLocalName()+":Error al desempacar el archivo hostsfilename");
        error = true;
        doDelete();}
myAgent.removeBehaviour(sifb);
}catch(Exception e){e.printStackTrace();
                error = true;
                doDelete();}
}
protected void handleOutOfSequence(ACLMessage msg){
System.err.println(getLocalName()+"Mensaje fuera de secuencia");
error=true;
```

```
doDelete();          }
protected void handleRefuse(ACLMessage refuse){
System.err.println(getLocalName()+":El clone ha refutado enviar el mensaje");
error=true;
doDelete();}
public int onEnd(){
if(!returnhome){
addBehaviour(new SNMPAgentTest(myAgent));
addBehaviour(new EmpaquetarFiles(myAgent, true));
addBehaviour(new ObtenerLocalizaciones(myAgent));
}
else{
try{
System.out.println("\nEL TEST A LA RED HA FINALIZADO
      SATISFACTORIAMENTE\n");
zipfile2 = new ZipFile(zipfilename);
entry2 = zipfile2.getEntry(hostsfilename);
if(entry2!=null){
is2 = new BufferedInputStream(zipfile2.getInputStream(entry2));
int readbytes = 0;
byte data[] = new byte[8192];
FileOutputStream fos2 = new FileOutputStream(entry2.getName());
while((readbytes = is2.read(data))!=-1){
fos2.write(data, 0, readbytes);}
is2.close();
fos2.close();}
File f5 = new File(dirfiles);
if(!f5.exists()){f5.mkdirs();}
if(f5.isDirectory()){

zipfile3 = new ZipFile(zipfilename);
Enumeration e = zipfile3.entries();
while(e.hasMoreElements()){
entry3 = (ZipEntry)e.nextElement();
is3 = new BufferedInputStream(zipfile3.getInputStream(entry3));
```

```
int readbytes = 0;
byte data[] = new byte[8192];
FileOutputStream fos3 = new FileOutputStream(dirfiles +File.separator +
        entry3.getName());
while((readbytes = is3.read(data))!=-1){fos3.write(data, 0, readbytes);}
fos3.close();
is3.close();}
File f6 = new File(dirfiles +File.separator + hostsfilename);
f6.delete();}
File f7 = new File(zipfilename);
f7.delete();
}
catch(Exception e){e.printStackTrace();
            error = true;
            System.err.println(getLocalName() +":Finalizado correctamente");}
finally{doDelete();}
        }
return 0;}
}
//Este behaviuor almacena el archivo remoto
class StoreIncomingFile extends SimpleBehaviour{
MessageTemplate mt;
ACLMessage msg;
FileOutputStream filefos;
byte[] filebytes;
public StoreIncomingFile(Agent a, FileOutputStream filefos){
super(a);
mt=MessageTemplate.and(MessageTemplate.MatchSender(new
        AID(getLocalName()+"clone",AID.ISLOCALNAME)),
        MessageTemplate.MatchPerformative(ACLMessage.UNKNOWN));
this.filefos = filefos;
}
public void action(){
msg = myAgent.receive(mt);
if(msg!=null){
```

```
try{
filebytes = (byte[])msg.getContentObject();
filefos.write(filebytes);
}catch(Exception e){e.printStackTrace();
            error = true;
            doDelete();
        }
}
//bloquear al behaviuor esperando otro mensaje
block();
        }
public boolean done(){
return false;
}
public int onEnd(){
try{
filefos.close();
}catch(Exception e){System.out.println(getLocalName()+":Error recibiendo el rchivo");
                        error = true;
                        doDelete();       }
return 1;}
}
protected void takeDown() {
try{
if(!cloned){DFService.deregister(this);}
if(!error){System.out.println(getLocalName()+":He finalizado mi trabajo");}
else{System.out.println(getLocalName()+":Finalizado debido a un error");}
}catch (Exception e){System.err.println("Error destruyendo al agente");}
                    }
}
```

```
//Código Java que realiza el proceso de empaquetamiento del AgenteMobilSNMP

package tesis.util;
import java.io.*;
import java.util.*;
import java.util.zip.*;

public class EmpaquetarDesempaquetar{
private boolean finished = false;
private String mode;
private String zipfilenamezip;
private ZipOutputStream zout;
private ZipFile zipfile;
public EmpaquetarDesempaquetar(String zipfilenamezip, String mode)throws Exception{
this.mode = mode;
this.zipfilenamezip = zipfilenamezip;
try{
this.iniciar();
}catch(Exception e){throw e;}
}
private void iniciar() throws Exception{
File f1 = new File(zipfilenamezip);
try{
if(mode.equals("add files") || mode.equals("new files") ){
zout = new ZipOutputStream (new FileOutputStream(zipfilenamezip + ".tmp"));
zout.setLevel(9);
if(mode.equals("add files")){
if(f1.canRead()){
        this.copiarFilesZip();}
        else{
                File f1t = new File(zipfilenamezip + ".tmp");
                f1t.delete();
                finished = true;
                Exception e = new Exception();
                throw e;}
```

```
            }
}
if (mode.equals("desempaquetar")){
zipfile = new ZipFile(zipfilenamezip);
            }
}catch(Exception e){
if(mode.equals("add files") || mode.equals("new files")){
File f2t = new File(zipfilenamezip + ".tmp");
f2t.delete();
            }
finished = true;
throw e;
        }
}
private void copiarFilesZip() throws Exception{
try{
ZipInputStream zis = new ZipInputStream(new BufferedInputStream(new
        FileInputStream(zipfilenamezip)));
ZipEntry entry = null;
while((entry = zis.getNextEntry())!=null){
this.addISPaquete(zis, entry.getName());}
zis.close();
}catch (Exception e){throw e;}}
private void addISPaquete(InputStream is, String paquetename)throws Exception{
try{
byte[] buffer = new byte[8192];
ZipEntry entry = new ZipEntry(paquetename);
zout.putNextEntry(entry);
int readbytes = 0;
while ((readbytes = is.read(buffer))!=-1){
zout.write(buffer, 0 , readbytes);
        }
zout.closeEntry();
}catch(Exception e){throw e;}}
```

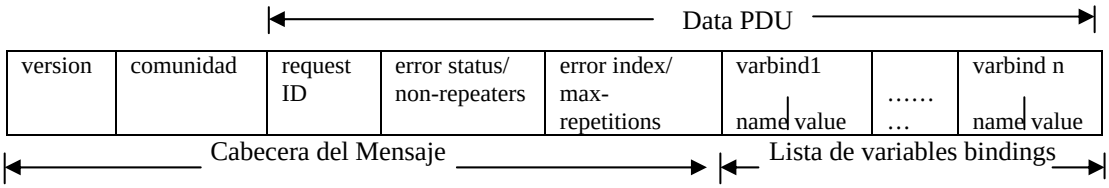
```
public void addPaquete(String empaquetararchivos, String archivoempaquetado) throws
        Exception, ZipException{
FileInputStream fis = null;
try{
if((mode.equals("add files") || mode.equals("new files")) && !finished){
fis = new FileInputStream (empaquetararchivos);
this.addISPaquete(fis, archivoempaquetado);
fis.close();
}
else{ZipException ze = new ZipException();
throw ze;}
}catch(Exception e){
fis.close();
File f3t = new File(zipfilenamezip + ".tmp");
f3t.delete();
throw e;}}
public void finDelEmpaquetamiento() throws Exception{
try{
if((mode.equals("add files") || mode.equals("new files")) && !finished){
zout.close();
File f4t = new File(zipfilenamezip + ".tmp");
File f2 = new File(zipfilenamezip);
f2.delete();
f4t.renameTo(f2);
finished = true;}
else{
finished = true;
File f5t = new File(zipfilenamezip + ".tmp");
f5t.delete();
ZipException ze = new ZipException();
throw ze;}
}catch(Exception e){
File f6t = new File(zipfilenamezip + ".tmp");
f6t.delete();
finished = true;
```

```
throw e;}}
public void unpackhostsfilename(String unpackdir) throws Exception{
try{
if(mode.equals("unpack") && !finished){
BufferedInputStream is = null;
ZipEntry entry;
Enumeration e = zipfile.entries();
while(e.hasMoreElements()){
entry = (ZipEntry)e.nextElement();
is = new BufferedInputStream(zipfile.getInputStream(entry));
int readbytes = 0;
byte[] data = new byte[8192];
FileOutputStream fos = new FileOutputStream(entry.getName());
while((readbytes = is.read(data))!=-1){
fos.write(data, 0 , readbytes);}
fos.close();
is.close();}}
}catch(Exception e){e.printStackTrace();}}
}
```

ANEXO D.

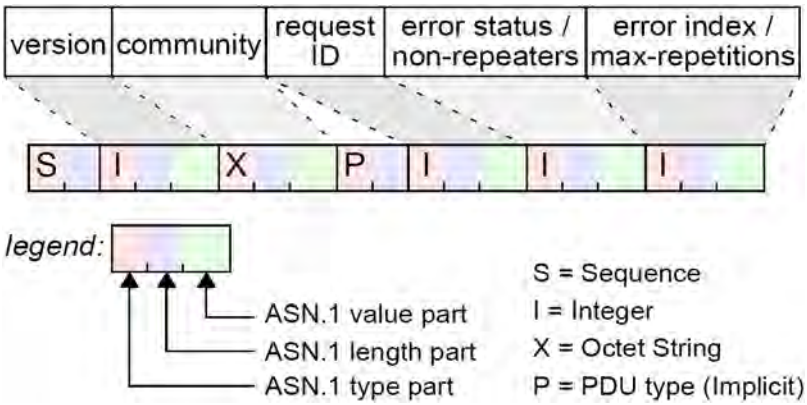
CODIFICACIÓN DE UN MENSAJE SNMP

La estructura de un mensaje SNMP (v1/v2) consta de dos partes [RFC1157 y RFC1901]: Una cabecera y una lista de variables bindings (enlazadas).



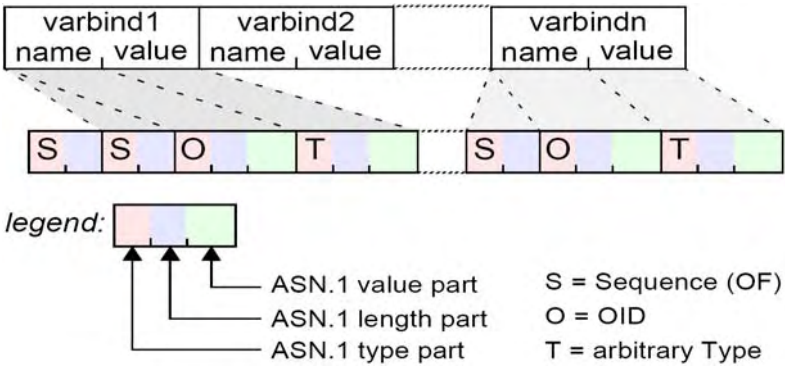
Anexo D: Figura1. Estructura de un mensaje SNMP (v1/v2) [ELLS].

Cada mensaje es definido en términos del constructor ASN.1, usando las reglas de codificación básica (BER) [Steffen02], para ser transmitido en una red alámbrica. Cada elemento de codificación BER consiste de tres partes: *type, length y value,* (Figura 2) [Pras04].



Anexo D: Figura2. Cabecera SNMP – Codificación BER [Pras04].

Desde que el mensaje entero SNMP es también definido como ASN.1 SEQUENCE, serán necesarios añadir dos octetos adicionales al código, al inicio del mensaje.



Anexo D: Figura 3. Lista de variables binding – codificación BER [Pras04].

S = Sequence (OF)
O = OID
T = Tipo arbitrario

1. ANCHO DEL MENSAJE SNMP

Número y tipo de octetos empleados en el mensaje SNMP:

INTEGER: Valor entre 1 y 5 octetos.

OCTET STRING: Depende de la longitud del String.

OBJECT IDENTIFIER: Depende del número de octetos de la longitud del OID menos uno. Por ejemplo: si un OID = .1.3.6.1.2.1.1.1.0 = 9, la parte *value* del codificador necesitará 8.

1.1 LONGITUD DE LA CABECERA

De las figuras 2 y 3, cuatro de los cinco campos son INTEGER y solo uno es de tipo OCTET STRING, entonces:

$$L_{cabecera} = 2 + 2 + 4.L_{INTEGER} + L_{OCTET\ STRING} \quad (1)$$

Desde que el mensaje completo SNMP es definido como una SEQUENCE ASN.1, dos octetos adicionales serán necesarios para el código y el inicio del mensaje.

Comunidad: public (6octetos)
private (7octetos).

Tipo: OCTET STRING

Longitud: 7 octetos

Version, Request ID, Error status, Error index:

Tipo: INTEGER

Longitud: 1octetos.

Reemplazando en la Formula 1:

$$L_{cabecera} \approx 4 + 4(\mathbf{2}+1) + \mathbf{2} + 7 \approx 25 \text{ octetos}$$

$$L_{cabecera} \approx 25 \text{ octetos} \quad (2)$$

1.2 LONGITUD DE LA LISTA VARBINDS

$$L_{varbin\ list} \approx 2 + n.(2 + L_{name} + L_{value}) \quad (3)$$

Donde:

n: Número de objetos MIB

1.2.1 LONGITUD DEL NOMBRE DE LOS OBJETOS (OID)

$$L_{name} \geq 2 + OID_{length} - 1 \quad (4)$$

Las variables utilizadas, en la aplicación de gestión SNMP son:

Variable MIB	OID	Longitud OID	Valor ASN.1
ifInOctets	.1.3.6.1.2.1.2.2.1.10.2	11	10
ifOutOctets	.1.3.6.1.2.1.2.2.1.16.2	11	10
ifSpeed	.1.3.6.1.2.1.2.2.1.5.2	11	10
sysUpTime	.1.3.6.1.2.1.1.3.0	9	8
ifInError	.1.3.6.1.2.1.2.2.1.14.2	11	10
ifInUcastPkts	.1.3.6.1.2.1.2.2.1.11.2	11	10
ifNUcastPkts	.1.3.6.1.2.1.2.2.1.12.2	11	10

Anexo D: Tabla 1. Variables utilizadas en la aplicación de gestión SNMP.

1.2.2 LONGITUD DEL VALOR DE LOS OBJETOS

$$L_{value} = 2 + S_{ObjectValue} \quad (5)$$

$S_{ObjectValue}$: Ancho del valor de los objetos (1 a 6 octetos) [Pras04]

El número de octetos necesarios para una codificación BER en un mensaje completo SNMP viene dado por:

$$L_{SNMP\ mensaje} = L_{cabecera} + L_{varbind\ list} \quad (6)$$

De las ecuaciones (2) y (3) reemplazando valores tenemos:

$$L_{SNMP\ mensaje} \approx 25 + 2 + n(2 + L_{name} + L_{value}) \quad (7)$$

De las ecuaciones (4) y (5) finalmente tenemos

$$L_{SNMP\ mensaje} \approx 27 + n(5 + OID_{length} + S_{Object\ value}) \quad (8)$$

De la ecuación (8) podemos deducir:

1. Para un mensaje de solicitud (request) vendría dado por:

$$L_{SNMP\ request} \approx 27 + n(5 + OID_{length}) \quad (9)$$

donde:

$S_{Object\ value} = 0$, para cada request, la codificación BER del valor del objeto requiere solo dos octetos (ASN.1 type = null, length = 0)

2. Para el mensaje de respuesta (réplica) con el contenido de la data viene dado por:

$$L_{SNMP\ response} \approx 27 + n(5 + OID_{length} + S_{Object\ value}) \quad (10)$$

Estableciendo valores y reemplazando en (9) y (10):

$OID_{length} = 11$ $n = 1$ $S_{Object\ value} = 0$ (request) $S_{Object\ value} = 5$ (replica)

- $L_{SNMP\ request} \approx 27 + 1(5 + 11) \approx 43$ octetos (11)

- $L_{SNMP\ response} \approx 27 + 1(5 + 11 + 5) \approx 48$ octetos (12)

- Dentro de la aplicación de gestión, se envía un mensaje de solicitud (request) por cada OID del MIB, de allí que n = 1.

GLOSARIO

1. **Calidad de Experiencia (QoE):** Captura de la experiencia de los usuarios relacionado a la QoS extremo a extremo.

2. **Calidad de Servicio (QoS):** Conjunto de parámetros que describe la calidad (por ejemplo ancho de banda, uso del buffer, prioridades, usos de CPU, y así sucesivamente) de un flujo especificado de datos.

3. **Middleware:** Es una capa de software que busca implementar servicios genéricos para el sistema distribuido como: identificación, autentificación, autorización, directorios y seguridad.

4. **Wsnmp32.**dll: Programa asociado al componente WinSNMP API que proporciona un conjunto de funciones para la codificación, decodificación, envío y recepción de mensajes.

5. **Mgmtapi.**dll: Programa asociado al componente Management API que proporciona un limitado conjunto de funciones que se pueden utilizar para el desarrollo de aplicaciones de gestión SNMP básicas de forma rápidas.